Развитие подхода к разработке тестов UniTESK

В. В. Кулямин, А. К. Петренко
{kuliamin,petrenko}@ispras.ru

Аннотация. В статье излагаются основные принципы, на которых основана технология UniTESK, предназначенная для создания тестов на основе формальных моделей. Суммируется опыт использования UniTESK в крупных проектах по разработке тестов для промышленных программных и аппаратных систем, включающих телекоммуникационные протоколы, базовые и стандартные интерфейсы операционных систем, блоки микропроцессоров. Дается обзор возможных направлений развития технологии в целях обеспечения ее большей масштабируемости.

Ключевые слова: тестирование на основе моделей, автоматизация тестирования, формальные спецификации, программные контракты, масштабируемость тестов.

1. Введение

Развитие технологий создания программного обеспечения (ПО) привело в последние десятилетия к беспрецедентному росту сложности создаваемых программных систем. Однако столь же разительного прогресса в технологиях обеспечения качества не наблюдается. Все более настоятельно ощущается необходимость в масштабируемых промышленно применимых технологиях обеспечения и контроля качества, которые позволяли бы разрабатывать надежные системы на современном уровне сложности.

Один из перспективных подходов к решению этой проблемы — использование максимально автоматизированных техник контроля качества совместно с компонентными технологиями, позволяющими строить иерархически скомпонованные программные системы высокой сложности. Примером подобной технологии является технология автоматизированного создания тестов UniTESK [1,2], разрабатываемая в ИСП РАН уже более 15 лет.

В данной статье дан обзор основных элементов технологии UniTESK, прослеживаются основные применения с момента создания первых прототипов поддерживающих ее инструментов, а также представлено несколько направлений развития технологии, связанных с потребностью в более масштабируемых тестах и инфраструктуре для их эффективного выполнения и удобного анализа их результатов.

2. Основные элементы технологии UniTESK

Тестированием называется проверка выполнения требований к системе при помощи наблюдения за ее работой в конечном наборе специально выбранных ситуаций [3]. поэтому любая технология построения тестов должна прежде всего обеспечивать решение следующих двух задач.

- Тесты должны проверять *требования* к проверяемой системе.
- Ситуации, используемые в тестах, должны обеспечивать определенную представительность по отношению ко всем возможным вариантам поведения проверяемой системы, иначе выводы о качестве системы, сделанные на основе проведенного тестирования, будут недостоверны. Данное свойство тестового набора принято называть полнотой тестирования и характеризовать с помощью выбираемых *критериев полноты*, задающих разбиения пространства всех возможных ситуаций на классы эквивалентности точки зрения возможных ошибок — если в одной из ситуаций данного класса возникает ошибка, она с большой вероятностью проявляется и в других ситуациях этого класса, если же система работает корректно, то и в других ситуациях того же класса это, скорее всего, так.

Кроме того, эффективность технологии построения тестов определяется трудоемкостью, или, наоборот, удобством, выполнения в ее рамках следующих действий.

- Разработка элементов тестов, создание которых невозможно (или крайне сложно) автоматизировать для широких классов систем. Примерами подобных элементов являются процедуры проверки требований или процедуры формирования определенных воздействий на систему через соответствующий интерфейс — они обычно специфичны для заданной системы или достаточно узкого семейства систем.
- Развертывание набора тестов перед выполнением в рабочем окружении.
- Выполнение набора тестов, включающее мониторинг важных для последующего анализа событий, поддержание проверяемой системы в рабочем состоянии, отслеживание обнаруживаемых сбоев и адекватное реагирование на них.
- Анализ результатов тестирования на предмет аккуратного выявления характеристик обнаруженных сбоев, их возможной локализации, а также полноты проведенного тестирования.
- Сопровождение тестового набора, включающее поддержку его работоспособности и развитие в соответствии с изменяющимися требованиями к проверяемой системе, а также изменениями в ее окружении и интерфейсах.

В рамках UniTESK для достижения достаточно широкой применимости технологии и уменьшения трудозатрат на создание тестов заданного уровня качества используются следующие решения.

- Предложена *унифицированная архитектура тестового набора* [1], определяющая набор компонентов тестов с четким разделением функций и заданными интерфейсами и нацеленная на реализацию в ее рамках большого многообразия тестов различных типов.
  Основные виды компонентов тестов: *тестовый оракул* [4], осуществляющий проверку соответствия наблюдаемого поведения отдельного компонента проверяемой системы требованиям; *тестовый сценарий* (см. ниже). При необходимости используются другие виды компонентов [5]: адаптеры, заглушки, генераторы данных, и т.п.
  Чтобы достичь высокой степени автоматизации, вся информация, которая может быть предоставлена только человеком, сконцентрирована в небольшом числе компонентов. Все остальные компоненты теста генерируются автоматически или используются во всех тестах в неизменном виде.
- Требования к проверяемой системе описываются в виде *программных контрактов* [6] с состоянием, состоящих из предусловий и постусловий интерфейсных операций и инвариантов компонентов, написанных в терминах работы с некоторым *модельным состоянием* компонента (т.е., структурой данных, позволяющей описать поведение операций, но, возможно, не совпадающей с той структурой данных, которая использована при его реализации). Программные контракты позволяют зафиксировать требования в виде, достаточно близком к их исходной формулировке (в частности, как в декларативном, так и в процедурном стиле), что облегчает дальнейшее развитие тестов при возникновении изменений в требованиях. Кроме того, программные контракты легко использовать для автоматической генерации тестовых оракулов.
  В ходе развития технологии контракты (и другие, создаваемые вручную компоненты) сначала описывались на специализированном формальном языке RSL (в рамках технологии KVEST [7], предшественника UniTESK), затем на расширениях широко используемых языков программирования [8,9], позже — с помощью набора библиотечных функций специального вида [10]. При этом, похоже, основную методическую роль при описании требований и построении тестов на их основе играет не сам используемый язык, а вкладываемая в него система понятий, соответствующих элементам архитектуры теста: проверяемый компонент, его интерфейс, интерфейсная операция, ее пред- и постусловия, модельное состояние компонента, его инвариант, пре-выражение в постусловии, вычисляемое до обращения к соответствующей операции. Кроме того,

существенно облегчает оформление контрактов использование конструкций, позволяющих учитывать особенности задания интерфейсов в языке, используемом для разработки тестируемой системы. Например, в языках, использующих исключения, они являются элементом интерфейса операций, и для адекватного описания постусловий необходимы специальные конструкции или библиотеки, позволяющие задавать требования к возникновению или отсутствию исключения, а также свойства возникающего исключения.

- Для описания асинхронного поведения и параллелизма используется специфическая техника [11-13]. Взаимодействие через асинхронный интерфейс представляется наборами событий, часть которых создается окружением проверяемой системы (это вызовы интерфейсных операций и сообщения, предназначенные системе), а все остальные — самой системой (это возвраты управления из интерфейсных операций и создаваемые системой сообщения). События описываются при помощи программных контрактов, каждый вид событий имеет пред- и постусловие, в частности, постусловия определяют, как меняется модельное состояние системы. При анализе взаимодействий теста с системой по нескольким каналам используется *семантика чередования*: наблюдаемый набор событий линейно упорядочивается так, чтобы в полученной цепочке сохранялся наблюдаемый порядок (события, получаемые по одному каналу, упорядочены), а также чтобы пред- и постусловия всех событий в цепочке выполнялись (т.е. модельное состояние, определяемое постусловием предшествующего события, удовлетворяло предусловию следующего). Если такую цепочку построить из данного набора событий невозможно, значит выявлено нарушение требований, зафиксированных в контрактах.

  Сам тест при этом разбивается на последовательность шагов, в рамках которых происходят отдельные этапы взаимодействия с тестируемой системой. На каждом шаге создается необходимый для воспроизводства определенной ситуации набор событий, после чего тест приостанавливается, пока не произойдут все события, генерируемые системой в ответ, т.е. пока новые события не перестанут возникать — *состояние покоя* (*quiescence*). Обычно наступление состояния покоя фиксируется по истечении некоторого времени после наступления последнего созданного системой события. Набор событий, произошедших на данном этапе, подвергается анализу с помощью описанной выше процедуры. При обнаружении линейного порядка событий, удовлетворяющего контрактам, считается, что текущее состояние проверяемой системы соответствует модельному состоянию в конце найденной цепочки. После этого тест переходит на следующий шаг, если такой имеется. Каждый отдельный

такой шаг считается нерасчленяемым (наблюдаемым образом) асинхронным воздействием на проверяемую систему.

- Для автоматического построения последовательности тестовых воздействий в тестах используется модель тестируемой системы (или одного/нескольких тестируемых компонентов) в виде *конечного автомата* (называемого *автоматом теста*). Тестовая последовательность строится как последовательность обращений к тестируемым операциям (или асинхронных воздействий в смысле, описанном в предыдущем пункте), соответствующая определенному пути в графе переходов автомата, например, обходу всех его переходов [14,15]. Автомат теста задается не полностью, а лишь в виде процедуры вычисления текущего состояния и набора действий (каждое действие может соответствовать некоторому набору создаваемых событий или цепочке вызовов интерфейсных операций), которые можно выполнить в произвольном состоянии. Используются также охранные условия, запрещающие выполнение определенных действий в некоторых состояниях. Такое описание называется *тестовым сценарием.* Действия могут быть параметризованы, и в этом случае для построения используемых при тестировании наборов значений параметров необходимо дополнительное указание источников или генераторов данных.
- В качестве критериев полноты тестирования в UniTESK применяются, в первую очередь, критерии покрытия кода контрактов, близкие к критериям покрытия требований. Критерии, устанавливаемые в качестве целей тестирования, служат основой при формировании структуры состояния тестового сценария и используемых тестовых данных, т.е., могут быть учтены лишь человеком-разработчиком тестов. Критерии полноты, отслеживаемые во время тестирования, должны быть преобразованы в критерий покрытия какого-либо кода [16] — для их контроля при выполнении тестов создаются специальные компоненты-мониторы покрытия, покрытие кода которых измеряется с помощью обычных средств.

В целом набор решений технологии UniTESK, как это подтверждается опытом ее применения в промышленной разработке, позволяет при помощи приемлемых трудозатрат создавать тесты высокого качества для весьма сложных систем [9,17], создание аналогичных тестов для которых традиционными методами требует гораздо больше ресурсов. Особенно заметным эффект применения технологии становится при длительном сопровождении тестового набора — за распределения информации, относящейся к разным аспектам тестов по различным компонентам, достигается существенное снижение затрат на сопровождение.

3. Опыт использования UniTESK

Технология UniTESK с момента создания прототипного набора поддерживающих ее инструментов применялась для разработки тестов для различного промышленного ПО.

- Базовые библиотеки и стандартные интерфейсы операционных систем (ОС). Такие проекты включают создание регрессионного набора тестов для ядра операционной системы Nortel Networks, выполнявшую еще, для помощи прототипа UniTESK, KVEST [7], создание тестового набора для отечественной ОС реального времени ОС 2000/ОС 3000, создание тестового набора OLVER [18] для базовой части стандарта Linux Standard Base [17] (LSB), включающей POSIX. В эту же категорию попадает разработка тестового набора для проверки соответствия стандарту ARINC 653 [19].
- Библиотеки времени выполнения языков программирования.
  Значительная часть базовых библиотек языка C покрывается проектом OLVER. Кроме того, технология UniTESK использовалась при разработке тестов для одной из реализаций инфраструктуры поддержки языка Java.
- Телекоммуникационные протоколы.
  Технология использовалась при создании тестовых наборов для проверки соответствия стандартам протоколов IPv6 [9], Mobile IPv6 [20], IPsec, а также таких протоколов, как SMTP, TFTP и пр.
- Имитационное тестирование моделей аппаратного обеспечения.
  UniTESK используется также для создания тестов для Verilog-моделей отдельных блоков и компонентов микропроцессоров и шин [21].

Разработанные при помощи UniTESK тестовые наборы обладают высокой сложностью, как из-за большого числа проверяемых интерфейсов, так и потому что они позволяют проводить аккуратное тестирование сложной функциональности. Самым масштабным набором тестов является OLVER: его код составляет около 500 тыс. строк на расширении языка C, тесты нацелены на проверку более 1500 функций LSB Core, разработанные контракты покрывают требования. размещенные на 6000 страниц текста стандартов LSB, POSIX, X/Open Curses, System V Interface Definition, ISO/IEC 9899 (стандарт языка программирования C). Разработанный для OLVER набор контрактных спецификаций на момент окончания проекта являлся самым объемным реально используемым набором формальных спецификаций в мировой практике (в дальнейшем аналогичный или несколько превосходящий по объему набор формальных моделей был создан в рамках проекта Microsoft по обеспечению открытости и интероперабельности протоколов взаимодействия, используемых в ее продуктах [22]). В большинстве приведенных примеров тестовых наборов использована специфическая для UniTESK техника тестирования асинхронных интерфейсов [11-13], позволяющая обнаруживать достаточно сложные ошибки.

Остановимся на наиболее интересных примерах использования технологии UniTESK и на полученных в при этом уроках (несколько более подробно об этом можно прочитать в [31]).

Первым применением UniTESK был поддержанный Microsoft Research проект по разработке тестового набора для реализации IPv6 [20]. Проект стартовал в 2000 году. Тогда UniTESK только начинал создаваться, поэтому пришлось использовать облегченную реализацию технологии на языке C — CTESK-light. Несмотря на нестабильность инструмента удалось построить эффективный тестовый набор, который нашел дефекты, ранее не обнаруженные другими тестовыми наборами. Это был первый опыт использования контрактных спецификаций для тестирования телекоммуникационных протоколов. Было показано, что контрактные спецификации в сочетании с техникой тестирования систем с асинхронными интерфейсами, разработанной в рамках UniTESK [13,37] позволяют строить эффективные тесты. Эти тесты выявили больше ошибок и требовали меньше усилий для создания и сопровождения при заданном уровне качества тестирования, чем тесты, построенные по традиционным технологиям. Вместе с тем, опыт тестирования протоколов показал, что для эффективной генерации последовательностей тестовых воздействий при тестировании протоколов полезно иметь и исполнимые модели.

Одним из первых опытов использования UniTESK для тестирования программных компонентов Java [8] через программный интерфейс (Application Programming Interface, API) стал проект тестирования одной из реализаций стандартной библиотеки поддержки времени исполнения Java. Разработка моделей и тестов не вызывала особых проблем, так как интерфейсы были хорошо документированы. Помимо интерфейсов на Java в системе также были интерфейсы на C++, но больших проблем и это не вызвало, поскольку архитектура тестов UniTESK предусматривает слой тестовых адаптеров. Более серьезные проблемы появились, когда началось собственно тестирование, в рамках которого генератор тестовой последовательности должен был работать на базе самой тестируемой системы, еще не стабильной в это время.

Одним из значимых примеров применения UniTESK на платформе Java является проект тестирования инфраструктуры распределенной информационной системы одного из крупных операторов мобильной связи, который продолжается и сейчас. Возможность формальной и строгой фиксации интерфейсов компонентов этой чрезвычайно большой и разнородной системы стала для заказчика самым главным преимуществом UniTESK по сравнению с другими инструментами функционального тестирования. В рамках проекта были формально специфицированы и протестированы сотни компонентов. Уже к концу первого года применения технологии положительный эффект проявился в ускорении сроков интеграции новых версий распределенной системы. Вместе с тем, вскрылась серьезная проблема. Если в предыдущих проектах применения UniTESK требования к

большей части интерфейсов определялись стандартами или другими тщательно разработанными документами, здесь уровень документирования часто оказывался недостаточным для построения консистентных спецификаций. Восстановление документации или требований к интерфейсам в системах такого размера оказывается практически неразрешимой задачей, что не позволяет использовать MBT в полном объеме. О путях решения этой проблемы будет кратко сказано в Заключении.

Самым крупным примером применения UniTESK стал проект OLVER (Open Linux VERification) [18], который проводился в 2005-2007 годах при поддержке Министерства образования и науки РФ. Целью проекта была формальная спецификация интерфейсов стандарта Linux Standard Base (LSB), точнее его центральная части LSB Core. В LSB Core входят наиболее важные библиотеки операционной системы Linux, которые в значительной части реализуют стандарт POSIX. Строгое описание стандарта LSB и наличие набора тестов, который мог бы качественно проверить соответствие разных реализаций библиотек Linux требованиям стандарта, являются необходимыми условиями для обеспечения переносимости приложений для Linux с одного дистрибутива на другой. Проблема переносимости приложений под Linux является крайне острой, поскольку сейчас доступно уже несколько сотен различных дистрибутивов. Результаты проекта опубликованы [18]. Были построены контрактные спецификации более чем 1500 интерфейсов на языке C. Инструментом моделирования и генерации тестов был выбран CTESK. В ходе проекта были выявлены проблемы в самих стандартах: LSB (ISO/IEC 23360) и The Single UNIX Specification, основную часть которого составляет стандарт POSIX.1 (он же IEEE Std 1003.1, он же ISO/IEC 9945, он же The Open Group Base Specifications Issue 6). Разработанный тестовый набор включен в пакет сертификационных тестов международного консорциума The Linux Foundatuion [38].

Опыт формализации интерфейсов большого индустриального стандарта и разработки тестового набора для него дал много полезных уроков. Одним из них стало понимание важности организации информационного и методического обеспечения такого проекта. Массив документации и исходных текстов библиотек Linux, особенно с учетом многочисленности версий и вариантов, связанных с различными аппаратными платформами, является огромным. Кроме того, в разработку собственно стандарта и в разработку его реализаций вовлечены тысячи людей, распределенных по всему миру. Из этого следует, что организация «документооборота» является одной из важнейших составляющих проектов такого калибра. В организационно-методическом плане мы столкнулись с тем, что обучение новых сотрудников и контроль за качеством спецификаций и тестов требует много усилий и при этом быстро достичь необходимого уровня профессионализма невозможно. То есть, масштабируемость проектов по использованию MBT в плане расширения числа участников проекта, задействованных в самой работе по

спецификации требований и отладке тестов, — это одна из самых сложных проблем, мешающая широкому внедрению MBT.

Одной из методических проблем является выбор уровня абстракции, на котором строится модель. Более абстрактные модели или разделение моделей на два-три слоя, отличающиеся уровнем абстракции, упрощают задачу переиспользования моделей и тестов, при этом общий размер системы возрастает, и ее сложность также растет. В долгосрочном плане выгоднее иметь многослойные модели, в краткосрочном — модели, близкие по уровню детальности к реализации, конечно, если реализация уже есть. Профессиональный опытный верификатор умеет находить баланс между абстрактным описанием поведения, например, файловой системы и особенностями, деталями интерфейса конкретной реализации файловой системы. В UniTESK имеется специальная поддержка разделения уровней абстракции. В частности, специфика конкретных интерфейсов может быть скрыта в слой адаптеров. Выбор баланса во многом определяется долгосрочными планами по использованию и развитию моделей и тестового набора. То есть, такого рода работа требует достаточно широкого кругозора, чего трудно требовать то обычных инженеров-тестировщиков.

Результаты проекта OLVER впоследствии были использованы в разработке тестового набора для российской операционной системы реального времени ОС 2000/3000 [19]. Эта система поддерживает две группы интерфейсов. Первая отвечает требованиям POSIX, вторая — требованиям ARINC 653, международного стандарта для критических встроенных систем. Выделение уровня адаптеров, разделяющего модельное и реализационное представление интерфейсов, заложенное в архитектуру UniTESK, существенно упростило повторное использование OLVER в данном проекте.

Одновременно с началом работ по OLVER были развернуты работы по применению UniTESK для симуляционного (или имитационного) тестирования моделей отдельных блоков микропроцессоров [21]. Полученные тесты использовались при разработке российских микропроцессоров с архитектурой MIPS и микропроцессоров с элементами VLIM/EPIC. Размер типовых блоков в таких микропроцессорах — несколько миллионов вентилей. Для целей спецификации и генерации тестов не потребовалось больших изменений в инструментах, за основу был взят CTESK. В техническом плане привязка CTESK к API на языке C не стала проблемой, так как большинство симуляторов, работающих с языками моделирования логики микропроцессоров (High Level Design languages, HLD), например, VHDL или Verilog, предоставляют удобный интерфейс для взаимодействия с программами на C. Несколько изменилась семантика предусловий в контрактных спецификациях, они стали описывать не столько разрешенную область входных данных, сколько условия возможности выполнения микрооперации на соответствующем такте, что, кстати, характерно и для семантики предусловий асинхронных событий при тестировании

параллелизма. Так же, как и в случае моделирования протоколов, выявилась потребность в использовании наряду с постусловиями, явных моделей поведения тестируемого устройства.

Как и в проектах по верификации программных систем одной из главных проблем, мешающих внедрению MBT при верификации аппаратного обеспечения (как и большинства других методов верификации), является отсутствие четких и детальных описаний функциональных требований к компонентам. Вместе с тем, ситуация в разработке микропроцессоров несколько лучше, так как в ее ходе принято наряду с HLD моделями строить и системные или архитектурные модели, описывающие семантику набора инструкций. Элементы таких архитектурных моделей можно использовать для восполнения недостающих знаний о поведении некоторых блоков микропроцессоров [39]. Хорошей новостью оказалось достаточно простое решение задачи распараллеливания выполнения теста на кластерах. Типичные размеры конечного автомата, который генерируется при выполнении теста одного сложного блока микропроцессора — это несколько миллионов узлов и десятки миллионов переходов. Оказалось, что генерация теста с помощью обхода неизвестного конечного автомата хорошо распараллеливается на кластерах до 200 узлов, с накладными расходами всего лишь 10-15%. При этом надо помнить, что при симуляционном тестировании высокоуровневой модели устройства основное время уходит на работу симулятора HDL, т.е. такая масштабируемость связана с возможностью запустить на каждом узле отдельный симулятор и выполнять параллельно много действий в различных состояниях. Масштабируемость распараллеливания тестирования систем других типов, скорее всего, будет не такой высокой.

Важно отметить задачи верификации, которые не удалось свести к моделированию на основе контрактных спецификаций, из-за чего они дали толчок для разработки новых методов MBT. В первую очередь следует упомянуть задачу тестирования компиляторов и задачу тестирования микропроцессора в целом, так называемый «core testing». В обоих случаях это задачи системного тестирования, где требуется подавать на вход большого «черного ящика» тестовые воздействия (в нашем случае это тестовые программы, которые подаются на вход компилятору или загружаются в память симулятора микропроцессора), но при этом интересно тестировать не все подряд, а лишь некоторые заданные режимы или заданную группу модулей компилятора или процессора. В случае тестирования компиляторов был разработан инструмент OTK, который использовался для тестирования оптимизирующих компиляторов Intel и Simulink [40,41]. Он позволяет нацеливаться на заданные виды оптимизаций. Для системного тестирования микропроцессоров был разработан инструмент MicroTESK [42-44]. Основной задачей этого инструмента была проверка разнообразных ситуаций, связанных с наиболее сложными подсистемами управления памяти: блоком трансляции адресов, кэшами разных уровней или блоком управления памятью в целом.

4. Направления развития технологии

Как показывает опыт применения UniTESK в больших проектах, развитие самого процесса разработки тестов в сторону повышения степени автоматизации возможно, однако, сводится, в основном, к проблемно-ориентированным решениям. Обычно можно спроектировать набор шаблонов тестов, использование которых может существенно облегчить создание тестов для конкретной системы или узкого семейства систем, но конкретный вид этих шаблонов становится ясен тогда, когда большая часть необходимых тестов уже разработана и отлажена. Предложить же аналогичное решение, способное снизить трудоемкость создания тестов во всех описанных выше примерах, крайне трудно.

Единственным исключением здесь является задача построения тестовых данных, которая, несмотря на значительный прогресс в развитии специализированных инструментов-решателей и рост числа их приложений для генерации тестов [23-26], не поддается прямолинейным решениям — практически все опубликованные подходы пригодны для автоматического построения только входных данных тестового воздействия, если те представляют собой даже достаточно объемные, но практически однородные наборы символов или чисел, и не принимают в расчет состояние тестируемой системы и сложные структуры входных данных (по сути также означающие необходимость создавать сложные объекты в различных состояниях).

Продвижение в сторону повышения степени автоматизации при построении тестовых данных в более общем случае возможно, скорее, на пути интеграции появившихся техник смешанного, символического и реального, исполнения тестируемого фрагмента кода [23-26] и автоматического обхода графа состояний сложного объекта за счет обращений к его интерфейсным методам.

Есть, также, еще ряд проблем, решение которых способно в достаточно общем случае повысить эффективность применения технологии UniTESK. Эти проблемы так или иначе связаны с масштабируемостью выполнения тестов и анализа их результатов. Дело в том, что технология уже позволяет создавать качественные тесты для достаточно сложной функциональности, но чисто количественное повышение их сложности, увеличивающее аккуратность тестирования, но не требующее существенных усилий от разработчиков (обычно за счет использования больших значений каких-то параметров, более сложных структур состояний, большего числа одновременно тестируемых операций, большего числа параллельно оказываемых воздействий и пр.) практически всегда приводит к значительному росту ресурсов, необходимых для выполнения таких тестов и анализа получаемых трасс, в т.ч. обнаруженных ошибок. С другой стороны, успешное повышение масштабируемости тестов может позволить решать задачи, в которых сейчас нужно тщательно выбирать структуру состояния и реализовывать достаточно сложные генераторы данных, за счет «грубой силы», используя простейшие решения, что также снизит общую трудоемкость создания тестов. Видимые

уже сейчас возможности по улучшению масштабируемости тестов следующие.

- Масштабируемость выполнения тестов может быть достигнута, прежде всего, за счет его *распараллеливания*. Сейчас чаще всего в тестах используются автоматы с сотнями состояний и тысячами переходов. Возможность выполнять тесты с миллионами состояний и десятками миллионов переходов за разумное время уже реализована за счет простейшего распределения копий тестируемой системы по узлам кластера [27] и использования общего состояния теста, что оказалось пригодным для тестирования сложных блоков аппаратного обеспечения. Еще больших показателей можно добиться, применяя специализированные параллельные алгоритмы исследования автоматов, не требующие общей памяти [28].
- Другой возможный источник повышения эффективности выполнения тестов — *редукция сериализаций*, выполняемых при поиске линейного упорядочения набора наблюдаемых событий на одном шаге теста асинхронной системы. В имеющихся инструментах строятся все возможные линейные порядки на данном наборе событий (отсекаются лишь те цепочки, которые являются продолжением невозможной из-за нарушения контрактов последовательности событий). Из-за возникающего комбинаторного взрыва (нужно еще учесть, что подобную сериализацию надо проводить на каждом использующем асинхронный интерфейс переходе некоторого автомата) за разумное время можно провести тестирование с использованием двух-трех видов асинхронных воздействий на каждом шаге (они обычно пополняются еще тремя-пятью асинхронными событиями, создаваемыми тестируемой системой), при возрастании этого числа до десятка время работы теста становится неприемлемым. Перспективными подходами к эффективному выполнению таких тестов являются использование техник динамической редукции частичных порядков [29,30] или выявления симметрий набора событий с помощью их символического выполнения.
- При выполнении сложных тестов, содержащих тысячи вызовов отдельных операций в сотнях различных состояний возникает достаточно объемная трасса (в упомянутых выше проектах трассы тестов могут достигать объема в десяток гигабайт). В случае обнаружения ошибок провести аккуратный анализ такого объема информации и четко выявить характеристики ошибки, чтобы можно было передать ее описание разработчикам проверяемой системы, нелегко. Еще хуже становится ситуация, если при работе одного теста выявляется несколько различных ошибок, причем каждая проявляется еще и в разных ситуациях десятки раз. Прекращение работы теста после каждого сбоя является не слишком эффективным решением —

такой тест надо будет перезапускать много раз, пока не будут устранены все проявляющиеся в ходе его работы ошибки, а каждое выполнение требует значительного времени.

Разумным выглядит такой сценарий работы сложных тестов.

- Если тест выполняется без обнаружения сбоев, после его работы остается только отчет о достигнутом тестовом покрытии и общее заключение об успешном выполнении.
- При обнаружении сбоя запоминается весь путь, приводящий к нему, после чего тест инициализирует тестируемую систему заново и продолжает работу, учитывая ранее полученную информацию (т.е., обнаруженные состояния и пройденные переходы в автоматной модели теста).
- После окончания работы теста с обнаруженными сбоями выполняется кластеризация этих сбоев по логическим признакам, описывающим соответствующие ситуации (эти признаки включают данные состояний, в которых происходят сбои, а также все данные выполняемых на пути до сбоя операций). Кластеризация необходима при обнаружении значительного числа (десятки) сбоев, поскольку каждый из них должен подвергаться тщательному анализу. Если удается автоматически сформировать разумные гипотезы об одинаковой природе различных сбоев и свести анализ всего лишь к нескольким ситуациям, трудоемкость его существенно снижается.
- После получения набора кластеров сбоев для каждого такого кластера (с учетом гипотезы об общих характеристиках попадающих в него ситуаций) строится наиболее короткий путь по графу автомата теста, повторяющий в конце этот сбой. Поскольку автомат теста является обычно лишь достаточно абстрактной моделью проверяемой системы, не учитывающей большое количество деталей ее работы, такой путь часто не соответствует кратчайшему пути по графу переходов автомата, и для его нахождения требуется перебор различных подпоследовательностей вызовов в исходной последовательности, приводящей к данному сбою.
- Таким образом, по итогам работы теста, обнаружившего ошибки, помимо отчета о достигнутом покрытии, остается список обнаруженных сбоев с их автоматически проведенной кластеризацией (сбои одного кластера, вероятно, являются проявлением одной и той же ошибки) и, для каждого кластера автоматически формируются набор условий, как можно точнее характеризующих попадающие в него ситуации, и кратчайшая последовательность вызовов операций, демонстрирующая такой сбой.

Тестировщику должна быть предоставлена возможность провести вручную перегруппировку обнаруженных сбоев с последующим автоматическим поиском минимальных подтверждающих тестов для новых групп.

Обозначенные в данном разделе направления развития технологии UniTESK находятся пока на стадии исследований. Проведены работы по проектированию алгоритмов параллельного обхода неизвестных графов без использования общей памяти, а также предложены некоторые техники построения минимальных тестовых последовательностей, демонстрирующих обнаруженную ошибку.

5. Заключение

В статье изложены основные элементы технологии UniTESK, обеспечивающие ее применимость для создания качественных тестов сложных систем и снижение трудоемкости сопровождения сложных тестовых наборов. Также суммирован опыт использования UniTESK в промышленных проектах разработки тестов, и представлено несколько направлений дальнейшего развития технологии.

Поскольку дальнейшее повышение сложности тестируемых систем требует более аккуратного тестирования, перспективы развития технологии связаны с обеспечением более эффективного выполнения объемных и сложных тестов, основанном на распараллеливании. Другое важное направление развития — повышение уровня автоматизации при анализе результатов тестов, поскольку анализ трасс из миллионов обращений к проверяемой системе слишком трудоемок для человека.

Работы по развитию технологии послужили отправной точкой как для создания специализированных средств верификации программ, так и для работ по развитию теории верификации. Текущее состояние этих исследований и разработок представлено в серии статей, которые в совокупности представляют собой достаточно полное описание направлений развития работ по созданию UniTESK.

Так в работе [32] описываются результаты развития теории построения тестов, проверяющих соответствие требованиям, в приложении к тестированию программных систем; в работе [33] рассматриваются результаты полученые в области тестирования операционных сиситем и новые задачи, которые предстоит решить в ближайшем будущем; в работе [34] рассматриваются вопросы использования UniTESK для автоматизации тестирования соответствия для телекоммуникационных протоколов; в работе [35] рассматриваются разработанные техники тестирования микропропроцессоров; в работе [36] рассматриваются вопросы построения интегрированной системы проектирования и верификации модульной авионики, в которой решение собственно задач тестирования увязываются и с другими задачами

жизненного цикла разработки ответветственного программного обеспечения, в частности с задачами анализа требований и систематического контроля выполнения требования на всех фазах жизненного цикла программ.

Литература

[1]. I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. *UniTesK Test Suite Architecture.* Proceedings of FME'2002, Kopenhagen, Denmark, LNCS 2391:77-88, Springer-Verlag, 2002.

[2]. В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. *Подход UniTesK к разработке тестов.* Программирование, 29(6):25-43, 2003.

[3]. ISO/IEC TR 19759 *Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Geneva, Switzerland: ISO, 2005.

[4]. L. Baresi, M. Young. *Test Oracles*. Tech. Report CIS-TR-01-02. 2001, http://www.cs.uoregon.edu/~michal/pubs/oracles.html.

[5]. В. В. Кулямин. *Организация сложных тестовых наборов*. Труды ИСП РАН, 17:9-24, 2009.

[6]. B. Meyer. *Applying Design by Contract*. IEEE Computer,25(10): 40-51, October 1992.

[7]. I. Bourdonov, A. Kossatchev, A. Petrenko, D. Galter. *KVEST: Automated Generation of Test Suites from Formal Specifications*. Proceedings of FM'99, Toulouse, France, LNCS 1708:608-621, Springer-Verlag, 1999.

[8]. I. B. Bourdonov, A. V. Demakov, A. A. Jarov, A. S. Kossatchev, V. V. Kuliamin, A. K. Petrenko, S. V. Zelenov. *Java Specification Extension for Automated Test Development.* Proceedings of PSI'2001, Novosibirsk, Russia, LNCS 2244:301-307, Springer-Verlag, 2001.

[9]. Г. В. Ключников, А. С. Косачев, Н. В. Пакулин, А. К. Петренко, В. З. Шнитман. *Применение формальных методов для тестирования реализации IPv6.* Труды ИСП РАН, 4:121-140, 2003.

[10]. В. В. Кулямин. *Компонентная архитектура среды для тестирования на основе моделей*. Программирование, 36(5):54-75, 2010.

[11]. V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin, A. S. Kossatchev, I. B. Bourdonov. *Integration of Functional and Timed Testing of Real-time and Concurrent Systems*. Proceedings of PSI'2003, Novosibirsk, Russia, LNCS 2890:450-461, Springer-Verlag, 2003.

[12]. V. Kuliamin, A. Petrenko, N. Pakoulin. *Practical Approach to Specification and Conformance Testing of Distributed Network Applications*. Proceedings of ISAS'2005, Berlin, Germany, LNCS 3694:68-83, Springer-Verlag, 2005.

[13]. А.В.Хорошилов. *Спецификация и тестирование компонентов с асинхронным интерфейсом*. Диссертация на соискание ученой степени к.ф.-м.н., Москва, 2006.

[14]. И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. *Использование конечных автоматов для тестирования программ*. Программирование, 26(2):61-73, 2000.

[15]. И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. *Неизбыточные алгоритмы обхода ориентированных графов: детерминированный случай.* Программирование, 29(5):59-69, 2003.

[16]. H. Zhu, P. A. V. Hall, J. H. R. May. *Software Unit Test Coverage and Adequacy*. ACM Computing Surveys, 29(4):366-427, Dec. 1997.

[17]. A. Grinevich, A. Khoroshilov, V. Kuliamin, D. Markovtsev, A. Petrenko, V. Rubanov. *Formal Methods in Industrial Software Standards Enforcement.* Proceedings of PSI'2006, Novosibirsk, Russia, LNCS 4378:459-469, 2006.

[18]. Проект OLVER, http://linuxtesting.org
[19]. A. Maksimov. *Requirements-Based Conformance Testing of ARINC 653 Real-Time Operating Systems*. Proceedings of Data Systems In Aerospace (DASIA) 2010, ESA SP-682, ISBN 978-92-9221-246-9, 2010.
[20]. Г. В. Ключников, А. С. Косачев, Н. В. Пакулин, А. К. Петренко, В. З. Шнитман. *Применение формальных методов для тестирования Mobile IPv6.* Сборник тезисов 2-й международной конференции «Интернет нового поколения», стр. 20-25, Ярославль, Россия, 2003.
[21]. В. П. Иванников, А. С. Камкин, А. С. Косачев, В. В. Кулямин, А. К. Петренко. *Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры*. Программирование, 33(5):47-61, 2007.
[22]. W. Grieskamp. *Microsoft's Protocol Documentation Program: A Success Story for Model-Based Testing*. Testing – Practice and Research Techniques. Lecture Notes in Computer Science, vol. 6303, p. 7, Springer, 2010.
[23]. P. Godefroid, N. Klarlund, K. Sen. *DART: Directed Automated Random Testing*. ACM SIGPLAN Notices — Proceedings of PLDI 2005, 40(6):213-223, 2005.
[24]. K. Sen, D. Marinov, G. Agha. *CUTE: a concolic unit testing engine for C*. Proceedings of ESES/FSE, pp. 263–272, 2005.
[25]. C. Cadar, V. Ganesh, P. Pawloski, D. Dill, D. Engler. *EXE: Automatically Generating Inputs of Death*. Proceedings of the 13-th International Conference on Computer and Communications Security CCS 2006, pp. 322-335.
[26]. C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball. *Feedback-Directed Random Test Generation*. Proc. of International Conference on Software Engineering, pp. 75-84, 2007.
[27]. И. Б. Бурдонов, С. Г. Грошев, А. В. Демаков, А. С. Камкин, А. С. Косачев, А. А. Сортов. *Параллельное тестирование больших автоматных моделей.* Вестник ННГУ, № 3, 2011, стр. 187-193.
[28]. И. Бурдонов, А. Косачев. *Обход неизвестного графа коллективом автоматов*. Труды Международной суперкомпьютерной конференции "Научный сервис в сети Интернет: все грани параллелизма". 2013, стр. 228-232.
[29]. C. Flanagan, P. Godefroid. *Dynamic Partial-Order Reduction for Model Checking Software*. ACM SIGPLAN Notices— Proceedings of POPL 2005, 40(1):110-121, 2005.
[30]. Y. Yang, X. Chen, G. Gopalakrishnan, R. Kirby. *Efficient Stateful Dynamic Partial Order Reduction*. Proceedings of SPIN 2008, LNCS 5156:288-305, Springer, 2008.
[31]. В. П. Иванников, А. К. Петренко, В. В.Кулямин, А. В. Максимов. *Опыт использования UniTESK как зеркало развития технологий тестирования на основе моделей.* Труды ИСП РАН, 23:207-218, 2013.
[32]. И.Б.Бурдонов, А.С.Косачев. *Развитие теории конформности: семантики, формальные модели, алгоритмы*. Труды ИСП РАН, 2014.
[33]. Герлиц Е.А., Кулямин В.В., Максимов А.В., Петренко А.К., Хорошилов А.В., Цыварев А.В. *Тестирование операционных систем*. Труды ИСП РАН, 2014.
[34]. Н.В.Пакулин, В.З.Шнитман. *Автоматизация тестирования соответствия для телекоммуникационных протоколов*. Труды ИСП РАН, 2014.
[35]. А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.Д. Татарников, М.М. Чупилко. *Средства функциональной верификации микропроцессоров*. Труды ИСП РАН, 2014.

[36]. С.В.Зеленов, А.К.Петренко, Н.В.Пакулин, А.А.Угненко, А.А.Хорошилов. *Инструментальные средства проектирования систем интегрированной модульной авионики*. Труды ИСП РАН, 2014.
[37]. Н. В. Пакулин, А. В. Хорошилов. *Разработка формальных моделей и тестирование соответствия для систем с асинхронными интерфейсами и телекоммуникационных протоколов*. Программирование, 33 (6), 26-55 (2007).
[38]. The Linux Foundation consortium. LSB certification test suite, http://ispras.linuxbase.org/index.php/LSB_Certification_System
[39]. Chupilko, M. M. *Developing Test Systems of Multi-Modules Hardware Designs*. Programming and Computer Software, 2012, 38(1):34-42, 2012.
[40]. Zelenov, S.V., Zelenova, S.A. *Model-Based Testing of Optimizing Compilers*. In: Proc. of the 19th IFIP TC6/WG6.1 International Conference on Testing of Software and Communicating Systems – 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007). LNCS, vol. 4581, pp. 365-377. Springer-Verlag, Berlin, 2007.
[41]. Zelenov, S.V., Silakov, D.V., Petrenko, A.K., Conrad, M., Fey I.: *Automatic Test Generation for Model-Based Code Generators*. In: IEEE ISoLA 2006 Second Intern. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Paphos, Cyprus, pp. 68-75, 2006.
[42]. Камкин, А.С.: *Метод автоматизации имитационного тестирования микропроцессоров конвейерной архитектуры на основе формальных спецификаций.* Диссертация на степень к.ф.-м.н., Москва, 2009.
[43]. Корныхин, Е.В.: *Метод автоматизации генерации тестовых программ для верификации MMU*. Диссертация на степень к.ф.-м.н., Москва, 2010.
[44]. Kamkin, A.S., Tatarnikov, A.: *MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors*. In: Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, 2012, Perm, Russia, 2012.

Evolution of UniTESK Test Development Technology

V. Kuliamin, A. Petrenko
Institute for System Programming, Russain Academy of Sciences, Moscow, Russia
{kuliamin,petrenko}@ispras.ru

Abstract. The paper considers almost 20-year evolution of UniTESK test development technology in ISP RAS and its future perspectives. It presents the basic principles of UniTESK: using formal software contracts as a base of test adequacy criteria and test oracle construction, uniform test suite architecture helping to organize various kinds of testing in one framework, interleaving semantics used to specify asynchronous interactions, and using extended FSM models derived from contracts and coverage criteria to generate test sequences automatically. The paper then summarizes experience of using UniTESK in large test development projects for software and hardware systems, including telecommunication protocols, basic and standard interfaces of operating systems, microprocessor units. Several directions of future technology development are depicted, all intended for higher scalability of test suites constructed: parallelization of test execution for large test suites, using more efficient techniques of asynchronous behavior analysis, automated merging of different bugs representing the same faults, automatic extraction of compact and most substantial bug description (including short demonstration scenarios) from huge data obtained as a result of long and complex automated test execution.

Keywords: model based testing, test automation, formal specifications, software contracts, test scalability.

References

[1]. I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. UniTesK Test Suite Architecture. Proceedings of FME'2002, Kopenhagen, Denmark, LNCS 2391:77-88, Springer-Verlag, 2002.

[2]. V. V. Kuliamin, A. K. Petrenko, A. S. Kossatchev, and I. B. Burdonov. The UniTesK Approach to Designing Test Suites. Programming and Computer Software, 29(6):310-322, 2003.

[3]. ISO/IEC TR 19759 Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK). Geneva, Switzerland: ISO, 2005.

[4]. L. Baresi, M. Young. Test Oracles. Tech. Report CIS-TR-01-02. 2001, http://www.cs.uoregon.edu/~michal/pubs/oracles.html.

[5]. V. V. Kuliamin. Organizatsiya slozhnykh testovykh naborov [Organization of complex test suites]. Trudy ISP RAN [Proceedings of ISP RAS], 17:9-24, 2009 (in Russian).

[6]. B. Meyer. Applying Design by Contract. IEEE Computer,25(10): 40-51, October 1992.

[7]. I. Bourdonov, A. Kossatchev, A. Petrenko, D. Galter. KVEST: Automated Generation of Test Suites from Formal Specifications. Proceedings of FM'99, Toulouse, France, LNCS 1708:608-621, Springer-Verlag, 1999.

[8]. I. B. Bourdonov, A. V. Demakov, A. A. Jarov, A. S. Kossatchev, V. V. Kuliamin, A. K. Petrenko, S. V. Zelenov. Java Specification Extension for Automated Test Development. Proceedings of PSI'2001, Novosibirsk, Russia, LNCS 2244:301-307, Springer-Verlag, 2001.
[9]. G. V. Kluchnikov, A. S. Kossatchev, N. V. Pakulin, A. K. Petrenko, V. Z. Shnitman. Primenenie formal'nykh metodov dlya testirovaniya realizatsii IPv6 [Using formal methods for IPv6 implementation testing]. Trudy ISP RAN [Proceedings of ISP RAS], 4:121-140, 2003 (in Russian).
[10]. V. V. Kuliamin. Component architecture of model-based testing environment. Programming and Computer Software, 36(5):289-305, 2010.
[11]. V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin, A. S. Kossatchev, I. B. Bourdonov. Integration of Functional and Timed Testing of Real-time and Concurrent Systems. Proceedings of PSI'2003, Novosibirsk, Russia, LNCS 2890:450-461, Springer-Verlag, 2003.
[12]. V. Kuliamin, A. Petrenko, N. Pakoulin. Practical Approach to Specification and Conformance Testing of Distributed Network Applications. Proceedings of ISAS'2005, Berlin, Germany, LNCS 3694:68-83, Springer-Verlag, 2005.
[13]. A. V. Khoroshilov. Spetsifikatsiya i testirovanie komponentov s asinkhronnym interfejsom [Specification and testing of components with asynchronous interface]. PhD Thesis, ISP RAS, Moscow, 2006 (in Russian).
[14]. I. B. Burdonov, A. S. Kossatchev, and V. V. Kulyamin. Application of Finite Automatons for Program Testing. Programming and Computer Software, 26(2):61-73, 2000.
[15]. I. B. Burdonov, A. S. Kossatchev, and V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, 29(5):245-258, 2003.
[16]. H. Zhu, P. A. V. Hall, J. H. R. May. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4):366-427, Dec. 1997.
[17]. A. Grinevich, A. Khoroshilov, V. Kuliamin, D. Markovtsev, A. Petrenko, V. Rubanov. Formal Methods in Industrial Software Standards Enforcement. Proceedings of PSI'2006, Novosibirsk, Russia, LNCS 4378:459-469, 2006.
[18]. OLVER project, http://linuxtesting.org
[19]. A. Maksimov. Requirements-Based Conformance Testing of ARINC 653 Real-Time Operating Systems. Proceedings of Data Systems In Aerospace (DASIA) 2010, ESA SP-682, ISBN 978-92-9221-246-9, 2010.
[20]. G. V. Kluchnikov, A. S. Kossatchev, N. V. Pakulin, A. K. Petrenko, V. Z. Shnitman. Primenenie formal'nykh metodov dlya testirovaniya Mobile IPv6 [Using formal methods for Mobile IPv6 conformance testing]. 2-nd International Conference "Next Generation Internet", pp. 20-25, Yaroslavl, Russia, 2003 (in Russian).
[21]. V. P. Ivannikov, A. S. Kamkin, A. S. Kossatchev, V. V. Kuliamin, and A. K. Petrenko. The use of contract specifications for representing requirements and for functional testing of hardware models. Programming and Computer Software, 33(5):272-282, 2007.
[22]. W. Grieskamp. Microsoft's Protocol Documentation Program: A Success Story for Model-Based Testing. Testing – Practice and Research Techniques. Lecture Notes in Computer Science, vol. 6303, p. 7, Springer, 2010.
[23]. P. Godefroid, N. Klarlund, K. Sen. DART: Directed Automated Random Testing. ACM SIGPLAN Notices — Proceedings of PLDI 2005, 40(6):213-223, 2005.

[24]. K. Sen, D. Marinov, G. Agha. CUTE: a concolic unit testing engine for C. Proceedings of ESES/FSE, pp. 263–272, 2005.
[25]. C. Cadar, V. Ganesh, P. Pawloski, D. Dill, D. Engler. EXE: Automatically Generating Inputs of Death. Proceedings of the 13-th International Conference on Computer and Communications Security CCS 2006, pp. 322-335.
[26]. C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball. Feedback-Directed Random Test Generation. Proc. of International Conference on Software Engineering, pp. 75-84, 2007.
[27]. I. B. Burdonov, S. G. Groshev, A. V. Demakov, A. S. Kamkin, A. S. Kossatchev, A. A. Sortov. Parallel'noe testirovanie bol'shikh avtomatnykh modelej [Parallel testing of large FSM models]. Vestnik NNGU [Bulletin of NNSU], № 3, 2011, pp. 187-193 (in Russian).
[28]. I. B. Burdonov, A. S. Kossatchev. Obkhod neizvestnogo grafa kollektivom avtomatov [Exploration of unknown graph by a set of autonata]. Proc. of International Conference "Science Service in the Internet: all facets of parallelism ". 2013, pp. 228-232 (in Russian).
[29]. C. Flanagan, P. Godefroid. Dynamic Partial-Order Reduction for Model Checking Software. ACM SIGPLAN Notices— Proceedings of POPL 2005, 40(1):110-121, 2005.
[30]. Y. Yang, X. Chen, G. Gopalakrishnan, R. Kirby. Efficient Stateful Dynamic Partial Order Reduction. Proceedings of SPIN 2008, LNCS 5156:288-305, Springer, 2008.
[31]. V. P. Ivannikov, A. K. Petrenko, V. V. Kuliamin, A. V. Maksimov. Opyt ispol'zovaniya UniTESK kak zerkalo razvitiya tekhnologij testirovaniya na osnove modelej [Experiences of UniTESK applications reflecting model based testing technology development]. Trudy ISP RAN [Proceedings of ISP RAS], 23:207-218, 2013 (in Russian).
[32]. I. B. Burdonov, A. S. Kossatchev. Razvitie teorii konformnosti: semantiki, formal'nye modeli, algoritmy [Development of conformance testing theory: models, semantics, algorithms]. Trudy ISP RAN [Proceedings of ISP RAS], 2014 (in Russian).
[33]. E. A. Gerlits, A. V. Maksimov, A. K. Petrenko, A. V. Khoroshilov. Testirovanie operatsionnykh sistem [Testing of operating systems]. Trudy ISP RAN [Proceedings of ISP RAS], 2014 (in Russian).
[34]. N. V. Pakulin, V. Z. Shnitman. Avtomatizatsiya testirovaniya sootvetstviya dlya telekommunikatsionnykh protokolov [Protocol conformance testing automation]. Trudy ISP RAN [Proceedings of ISP RAS], 2014 (in Russian).
[35]. A. S. Kamkin, A. M. Kotsyniak, S. A. Smolov, A. D. Tatarnikov, M. M. Chupilko. Sredstva funktsional'noj verifikatsii mikroprotsessorov [Microprocessor functional verification tools and methods]. Trudy ISP RAN [Proceedings of ISP RAS], 2014 (in Russian).
[36]. S. V. Zelenov, A. K. Petrenko, N. V. Pakulin, A. A. Ugnenko, A. V. Khoroshilov. Instrumental'nye sredstva proektirovaniya sistem integrirovannoj modul'noj avioniki [Integrated modular avionics system design tools]. Trudy ISP RAN [Proceedings of ISP RAS], 2014 (in Russian).
[37]. N. V. Pakulin, A. V. Khoroshilov. Development of Formal Models and Conformance Testing for Systems with Asynchronous Interfaces and Telecommunications Protocols. Programming and Computer Software, 33(6):316-335, 2007.
[38]. The Linux Foundation consortium. LSB certification test suite, http://ispras.linuxbase.org/index.php/LSB_Certification_System
[39]. M. M. Chupilko, Developing Test Systems of Multi-Modules Hardware Designs. Programming and Computer Software, 2012, 38(1):34-42, 2012.

[40]. S. V. Zelenov, S. A. Zelenova. Model-Based Testing of Optimizing Compilers. In: Proc. of the 19th IFIP TC6/WG6.1 International Conference on Testing of Software and Communicating Systems – 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007). LNCS, vol. 4581, pp. 365-377. Springer-Verlag, Berlin, 2007.

[41]. S. V. Zelenov, D. V. Silakov, A. K. Petrenko, M. Conrad, I. Fey. Automatic Test Generation for Model-Based Code Generators. In: IEEE ISoLA 2006 Second Intern. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Paphos, Cyprus, pp. 68-75, 2006.

[42]. A. S. Kamkin Metod avtomatizatsii imitatsionnogo testirovaniya mikroprotsessorov konvejernoj arkhitektury na osnove formal'nykh spetsifikatsij [Pipeline microprocessor simulation testing automation based on formal specifications]. PhD Thesis, ISP RAS, Moscow, 2009 (in Russian).

[43]. E. V. Kornykhin. Metod avtomatizatsii generatsii testovykh programm dlya verifikatsii MMU [Automated generation of etst programs for MMU verification]. PhD Thesis, ISP RAS, Moscow, 2010 (in Russian).

[44]. A. S. Kamkin, A. D. Tatarnikov. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. In: Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, 2012, Perm, Russia, 2012.