Формализация интерфейсных стандартов и автоматическое построение тестов соответствия

В. В. Кулямин, А. К. Петренко, В. В. Рубанов, А. В. Хорошилов

Институт системного программирования РАН (ИСП РАН),
Б. Коммунистическая, 25, Москва, Россия
E-mail: {kuliamin, petrenko,vrub,khoroshilov}@ispras.ru

Аннотация

В статье описывается подход к построению инфраструктуры эффективного использования стандартов на программные интерфейсы (API). Предлагаемый подход основан на описании требований таких стандартов в виде формальных спецификаций и автоматическом построении тестов для проверки соответствия этим требованиям. В рамках этого подхода предлагается технологическая поддержка для решения ряда возникающих инженерных и организационных задач, что позволяет использовать его для сложных промышленных стандартов программного обеспечения. Этот тезис иллюстрируется использованием описанного метода для формализации стандарта Linux Standard Base Core 3.1 [5], промышленного стандарта на интерфейсы базовых библиотек операционной системы Linux.

Введение

Масштабность современных программных систем приводит к тому, что такие системы строятся из многих компонентов, разрабатываемых в разных организациях огромным количеством людей. Для построения работоспособных систем на основе такого подхода необходимо уметь решать задачи обеспечения совместимости между различными их компонентами и надежности систем в целом. Лучшее решение, которое было найдено на этом пути — выработка и соблюдение *стандартов на программные интерфейсы* между отдельными компонентами.

Идея, лежащая в основе использования интерфейсных стандартов, проста — настаивая на их соблюдении, мы обеспечиваем компонентам, созданным разными разработчиками, возможность взаимодействовать через стандартизованные интерфейсы. Таким образом, их совместимость обеспечивается без введения чересчур жестких ограничений на возможные реализации, что ограничило бы как творчество отдельных разработчиков, так и инновационный потенциал компаний-поставщиков ПО. Этот подход хорошо работает, если стандарт определяет функциональность, реализуемую фиксируемым им интерфейсом, достаточно точно и недвусмысленно.

Однако, тексты многих современных стандартов, описывающих требования к программным интерфейсам, далеко не так точны. Конечно, некоторые из действующих стандартов на интерфейсы ПО или на телекоммуникационные протоколы появились еще 20-25 лет назад и неоднократно пересматривались, что делало каждую их новую версию более строгой и непротиворечивой, и большинство проблем первых выпусков уже устранено. Однако неясности и противоречия постоянно вносятся в новых добавлениях, которые так и будут появляться в связи с постоянным развитием технологий.

Выход из этого замкнутого круга многие специалисты по программной инженерии видят в *формализации требований стандартов.* Формализация большого текста, конечно, требует приложения колоссальных усилий, но она может быть проведена не в один прием, а постепенно, инкрементально. Никто также не ожидает, что все неточности удастся удалить из стандарта одним махом. Начав процесс формализации,

мы, по крайней мере, получаем информацию об имеющихся реальных проблемах стандарта и можем предлагать и анализировать различные варианты их решения.

Формализация делает стандарт более точным, но одной формализации все же недостаточно для эффективного внедрения и использования стандарта на практике. Разработчики ПО нуждаются во вспомогательных инструментах, которые позволят им простым способом убедиться в соответствии или несоответствии той или иной системы стандарту. Поэтому, формальное описание требований стандарта должно дополняться набором тестов, которые проверяют соответствие этим требованиям. Основные положения по формализации стандартов и разработке тестов на соответствие представлены в стандартах ISO 9646 [1] и ITU-T Z.500 [2].

Продуманная методика делает тестирование на соответствие стандарту более строгим и тем самым обеспечивает более высокое качество ПО, реализующего его. Однако практическое использование наборов тестов на соответствие порождает ряд инженерных и организационных вопросов, без решения которых невозможно перенести применение этой методики для небольших примеров на современные стандарты, описывающие очень сложные системы. Можно отметить следующие аспекты, требующие особого внимания.

- *Прослеживаемость требований.* Для разработчиков ПО и их руководителей подлинным источником требований является сам текст стандарта. Формальные спецификации же — это какой-то дополнительный документ, который должен явно продемонстрировать свою связь со стандартом. То же самое относится и к тестам, полученным из формальных спецификаций — они должны соотноситься с требованиями, описанными в каких-то разделах стандарта. Прослеживаемость требований стандарта в тестах является практической потребностью всякой осмысленной деятельности по их реализации.

- *Покомпонентное рассмотрение стандарта.* Промышленные стандарты иногда очень сложны и часто имеют большой объем, так же как и промышленные программные системы. Для того чтобы адекватно анализировать их, необходимы какие-то техники декомпозиции. Формализм, используемый для спецификаций, должен позволять выделять в стандарте компоненты разных уровней, и анализировать их по отдельности (разнося по времени и/или по ресурсам). Должна поддерживаться также и последующая интеграция таких компонентов в единое целое.

- *Управление изменениями.* Стандарты, как и их реализации, не пребывают в неизменном виде — они постоянно изменяются под давлением общего технологического прогресса. Необходима адекватная поддержка изменений в используемом формализме, методах построения тестов и инструментах. Ее отсутствие быстро сделает полученные спецификации и тесты практически бесполезными.

- *Управление конфигурациями.* Очень часто стандарты, используемые на практике, описывают системы с большим числом параметров–конфигурационных опций, значительно влияющих на поддерживаемую функциональность. Возможность выбора конфигурации и проверки только относящихся к ней требований также должна быть заложена в спецификации, тесты и методику их разработки.

Все перечисленные проблемы должны иметь удобные решения в рамках технологии построения и эксплуатации инфраструктуры для эффективного использования стандартов. В данной статье рассказывается о подходе, претендующем на роль такой технологии и основанном на методе автоматизированной разработки тестов UniTESK [10-13, 27], созданном в ИСП РАН. Основные идеи и техники этого подхода

рассматриваются в следующем разделе статьи. Второй раздел представляет предварительные результаты его применения к Linux Standard Base Core 3.1 [5], промышленному стандарту на интерфейсы основных системных библиотек операционной системы Linux. Проект выполняется в рамках Центра верификации ОС Linux при ИСП РАН. В последних двух разделах статьи содержится краткий обзор других подходов к построению тестов на соответствие стандартам и заключение, формулирующее основные результаты и перечисляющее направления возможного развития предложенного подхода.

1 Формализация стандартов и тесты соответствия

Основные трудности, возникающие при попытке точно представить требования индустриальных стандартов, связаны с их неформальной природой. Главные цели такого стандарта — зафиксировать консенсус ведущих производителей относительно функциональности рассматриваемых систем и обеспечить разработчиков реализаций стандарта и приложений, работающих на основе этих реализаций, справочной информацией и руководством по использованию описанных функций. Форма и язык стандарта выбираются так, чтобы достигать этих целей. Чаще всего исходный текст стандарта представляется на естественном языке.

Стандарты на программные интерфейсы обычно состоят из двух частей: обоснования (rationale), представляющего целостную картину основных концепций и функций в рамках данного стандарта, и справочника (reference), описывающего каждый элемент интерфейса отдельно. В дополнение к элементам интерфейса (типам данных, функциям, константам и глобальным переменным) справочник может описывать и более крупные компоненты: подсистемы, заголовочные файлы и пр. Отдельные разделы справочника могут ссылаться друг на друга и содержать части друг друга.

Рассматриваемый подход к формализации стандарта и созданию соответствующих тестов включает несколько видов деятельности.

1. ***Декомпозиция стандарта*** – группировка интерфейсных элементов, описанных в стандарте, на логически связанные группы для организации дальнейшей работы в рамках этих групп, разнося их по времени и/или по ресурсам.
2. ***Выделение требований*** – определение всех требований стандарта к описываемым им элементам интерфейса, которые могут быть проверены. Основной результат этой работы – *каталог элементарных требований стандарта*.
3. ***Разработка спецификаций*** – преобразование выделенных неформальных требований в машинно-читаемые описания (*формальные спецификации*) на специализированном языке. Такие спецификации задают правила проверки корректности работы интерфейса при заданных входных аргументах в заданном состоянии системы.
4. ***Разработка тестов*** – создание тестовых сценариев, контролирующих многообразие воздействий на тестовую систему, и автоматическая генерация тестов на основе разработанных формальных спецификаций и этих сценариев.

В результате первых трех стадий формируются следующие результаты (возникающие при этом проблемы и предлагаемые методы их решения подробнее описаны в работах [3]-[4]):

- ***Каталог требований***. Он состоит из списка проверяемых требований, налагаемых стандартом, и привязывает каждое требование к соответствующему месту в тексте стандарта. Одно требование обычно соответствует логически

замкнутому фрагменту текста, выражающему одно ограничение на элемент интерфейса или элемент данных. Каждое требование имеет уникальный идентификатор. Каталог требований позволяет в дальнейшем оценивать адекватность тестирования в терминах исходного текста стандарта.

- ***Размеченный текст стандарта***. Это исходный текст стандарта, в котором наглядным образом, например цветом, выделены места, соответствующие требованиям из каталога. Размеченный текст позволяет проверить полноту выполненного анализа текста стандарта и убедиться в том, что ни одно требование не пропущено.

- ***Дефекты стандарта и замечания к его тексту***. Это список обнаруженных двусмысленностей, несоответствий между утверждениями в различных частях стандарта, ненамеренно неясных и неточных ограничений, неполных описаний функциональности и т.д. Этот список вместе с рекомендациями по устранению передается группе разработчиков стандарта, и позволяет сделать следующие его версии более точными и непротиворечивыми.

- ***Формальные спецификации*** всех элементов интерфейса. Код спецификаций размечается, чтобы указать связь между описанными формальными ограничениями и соответствующими им требованиями из каталога требований.

- ***Конфигурационная система стандарта***. Эта система состоит из набора конфигурационных параметров, как объявленных в стандарте, так и дополнительно введенных разработчиками спецификаций, зависимостей между ними, а также связей между ними, элементами интерфейса и проверяемыми ограничениями. Некоторые конфигурационные опции, представленные как значения параметров, управляют набором проверяемых требований. Другие могут говорить о присутствии или отсутствии определенной функциональности в системе. Третьи могут влиять на возможные коды ошибок, возвращаемые операциями.

  Пример первого случая можно увидеть в описании функции `pthread_create()` из стандарта POSIX: «Если макрос `_POSIX_THREAD_CPUTIME` определен, то новый поток должен иметь доступ к часам процессорного времени, и начальное значение этих часов должно быть выставлено в ноль» ("If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock accessible, and the initial value of this clock shall be set to zero").

В основе процесса разработки тестов на соответствие стандарту лежит технология UniTESK, разработанная в ИСП РАН на базе многолетнего опыта проектов по тестированию различного ПО. Основные идеи этой технологии заключаются в следующем.

- Функциональные требования к поведению тестируемой системы представляются в виде *контрактных спецификаций*. Контракт для отдельного компонента состоит из пред- и постусловий для всех его операций и асинхронных событий, которые он может создавать, и инвариантов, определяющих ограничения целостности его внутренних данных. Фактически контрактные спецификации задают универсальные правила проверки корректности поведения интерфейса для любых допустимых входных параметров и состояний системы.

- *Критерии тестового покрытия* определяются на основе структуры постусловий операций отдельного компонента или нескольких компонентов. Примерами таких критериев являются покрытие ветвей в коде постусловия и

покрытие дизъюнктов из дизъюнктивной нормальной формы условий этих же ветвлений. Есть возможность вводить произвольные критерии, описывая дополнительные ситуации, в которых работу системы необходимо проверить.

- *Тестовый сценарий* создается для достижения определённого покрытия заданной группы операций. Такой сценарий может описывать как отдельные воздействия на систему с определенными наборами параметров, так и определять конечный автомат, моделирующий поведение тестируемого компонента таким образом, что выполнение всех его переходов гарантирует покрытие 100% ситуаций в соответствии с выбранным критерием. Автомат задается при помощи функции вычисления состояния и набора допустимых действий в произвольном состоянии. Этот набор действий зависит от данных состояния. Каждое действие соответствует вызову одного из интерфейсов с некоторыми аргументами.
- *Тестовый запускаемый комплекс* формируется из тестов, автоматически сгенерированных на основе контрактных спецификаций и тестовых сценариев, и *конфигурационной системы* тестов. Эта система включает конфигурационную систему стандарта (см. выше) и дополнительные параметры, управляющие работой тестов. Разные значения этих параметров соответствуют различным критериям покрытия, разным наборам тестовых сценариев и т.п. Хорошая конфигурационная система делает тестовый комплекс применимым в любых условиях, где может функционировать реализация рассматриваемого стандарта.

2 Применения описанного подхода

Представленный выше подход был успешно использован для формализации и создания наборов тестов для частей стандартов на протоколы IPv6 и IPMP2 [11,14].

Более масштабным применением этого подхода стал проект Центра верификации ОС Linux [15] по формализации Linux Standard Base Core 3.1 [5] и разработке тестового набора для проверки соответствия ему. Проект был назван OLVER — Open Linux VERification. Цель проекта — создать формальные спецификации и соответствующий тестовый набор для 1532 функций основных системных библиотек Linux, перечисленных в разделах III и IV указанного стандарта (они описывают базовые библиотеки и библиотеки утилит).

Стандарт LSB Core задает требования ко многим из функций, ссылаясь на уже существующие стандарты. Непосредственно в тексте LSB описывается лишь 15% интерфейсов, большинство — 60% функций — определяются в стандарте Single UNIX Specification [6], который включает текущую версию POSIX, а остальные — в таких спецификациях как X/Open Curses [16], System V Interface Definition [17] и ISO/IEC 9899 (стандарт языка C) [18]. LSB не включает в себя все эти стандарты, а ссылается лишь на их части, касающиеся описания требований для отдельных функций. Всего, вместе со стандартами, на которые он ссылается, LSB Core 3.1 состоит из более чем 6000 страниц текста.

В начале проекта 1532 функции LSB Core были разбиты на 170 групп функционально связанных операций, которые в свою очередь группируются в большие подсистемы в соответствии с общей функциональностью — потоки, межпроцессное взаимодействие, файловая система, управление динамической памятью, математические функции, и т.д.

3 октября 2006 года выпущена версия 0.8 тестового набора OLVER - на эту дату проанализирован текст стандарта для 1376 функций, выделено около 14000 элементарных требований, при этом к некоторым функциям предъявляется всего лишь несколько требований, в то время как к другим — несколько десятков. Из этих

функций для 1155 разработаны формальные спецификации и тесты. Полученный набор спецификаций и тестов содержит около 500 тысяч строк кода. При этом важным «побочным эффектом» работы является обнаружение проблем в текущем тексте стандарта (нечеткие, двусмысленные, противоречивые или ошибочные места) и формирование предложений разработчикам стандарта по их устранению. В настоящее время в рамках проекта опубликовано 67 таких замечаний. Текущее состояние и результаты проекта доступны на сайте [15].

Промежуточный показатель по количеству покрываемых тестами OLVER интерфейсов LSB Core 3.1 уже сейчас превышает показатели известных тестовых наборов, среди которых стоит отметить:

- регрессионные тесты для библиотеки glibc [19], созданные ее разработчиками;
- тесты LTP [20], созданные при поддержке IBM специально для тестирования системных интерфейсов Linux;
- официальный тестовый набор LSB Core [21] от Free Standards Group.

Сравнительные показатели этих тестов приведены в Таблице 1.

	LTP [20]	LSB TS [19]	glibc TS [21]	OLVER [15]
Число покрываемых функций LSB Core 3.1	566	643	873	1155

Таблица 1. Сравнение количества тестируемых функций.

Полученные предварительные результаты позволяют надеяться, что рассмотренный подход может успешно применяться для проектов такого масштаба.

3 Другие подходы к построению тестов на соответствие стандартам

Наиболее глубокие результаты в области формализации стандартов и методик разработки тестов на основе формальных спецификаций получены применительно к телекоммуникационным протоколам. Общие положения подхода к формальному тестированию, хорошо представленны в работах Берно (Bernot) [7], Бринксмы (Brinksma) и Тритманса (Tretmans) [8,9] и описанны в телекоммуникационных стандартах ISO 9646 [1] и ITU-T Z.500 [2].

Подход к формальному тестированию в телекоммуникации имеет много общего с представленным в данной статье. Различия связаны, в основном, с необходимостью иметь дело со стандартами большего объема, такими как стандарты POSIX или LSB на интерфейсы библиотек операционных систем. Большой размер стандартов и соответствующих спецификаций приводит к практической невозможности использовать как полные тестовые наборы в терминах работы [9], поскольку они бесконечны, так и выбор набора целей тестирования «на глаз», который является одним из традиционных шагов разработки тестов для телекоммуникационных протоколов. Вместо этого используется более систематическое выделение отдельных требований, понятие критерия тестового покрытия, выбор критерия, ориентированного на учет всех выявленных требований, и генерация тестов, нацеленная на достижение высоких показателей покрытия по выбранному критерию.

Использование контрактных спецификаций также способствует большей практичности и масштабируемости нашего подхода. Программные контракты, с одной стороны, позволяют провести декомпозицию большой системы на более обозримые компоненты, что труднее сделать, используя автоматы или системы переходов, лежащие в основе

традиционного формального тестирования. С другой стороны, пред- и постусловия лучше подходят для описания недетерминированного поведения, которое довольно часто вынужден фиксировать стандарт при наличии нескольких возможностей реализовать одну и ту же абстрактную функциональность.

Статья [22] представляет другую попытку формализации стандарта на примере IEEE 1003.5 — POSIX Ada Language Interfaces (интерфейсы POSIX для языка Ada). В рамках описываемого в ней метода на базе требований стандарта сразу строились формальные описания проверяющих их тестов, без промежуточных спецификаций самих требований. Такой метод кажется нам принципиально мало отличающимся от традиционной ручной разработки тестов, при которой разработчик теста читает стандарт, придумывает на основе прочитанного тесты и записывает их в виде некоторых программ.

Более близкий к рассматриваемому в данной статье подход используется в работе [23], где представлены результаты практического использования инструмента автоматизированного создания тестов GOTCHA-TCBeans. Этот инструмент использовался, в частности, для построения тестов для функции `fcntl()` стандарта POSIX. По ряду идей, касающихся использования формальных техник для тестирования, представленный в [23] подход очень похож на используемый нами, хотя он более сфокусирован на технике создания формальных моделей, чем на обеспечении их адекватности и поддержке прослеживаемости связей между формальными спецификациями и исходным текстом стандарта. Кроме того, используемые в [23] формальные спецификация являются описаниями конечных автоматов на специализированном языке Murφ, а не программными контрактами, как в нашем подходе.

Существует ряд аналогичных работ по разработке тестовых наборов для проверки соответствия стандартам интерфейсов операционных систем. Наиболее известные стандарты в этой области это IEEE Std 1003.1, или POSIX [6], и LSB [5]. Имеются и наборы тестов на соответствие этим стандартам — это сертификационные тесты POSIX от Open Group [24], открытый проект Open POSIX Test Suite [25], и официальные сертификационные тесты на соответствие LSB [21] от Free Standards Group.

Все эти проекты используют схожие технологии выделения требований из текста стандарта и создания соответствующего каталога требований. После этого тесты разрабатываются вручную на языке C с применением подхода «один тест на одно требование». Они не используют формализацию требований и автоматическую генерацию тестов.

Стоит отметить, что использование подхода «один тест на одно требование» создает предпосылки для укрупнения требований при их выделении, так как велик соблазн проверить как можно больше в одном тесте. К примеру, в тестах Open POSIX мы обнаружили требования, которые включают в себя десяток утверждений, которые в проекте OLVER выделяются в отдельные требования. Такое укрупнение требований может приводить к тому, что некоторые утверждения стандарта упускаются из виду и не проверяются, в то время как крупное интегральное требование, в которое они входят, считается протестированным. В результате построенный на основе таких требований отчет об их покрытии может искажать реальное положение дел, утверждая, что все требования стандарта проверены.

Кроме того, автоматическая генерация тестов из спецификаций, используемая в нашем подходе, делает тестовый набор более управляемым, позволяя описывать сами требования стандарта в одном месте, а технику, используемую для их проверки, и тестовые данные — в другом. Это значительно облегчает внесение изменений в тесты

при развитии стандарта или их адаптации под требования специфической предметной области.

Заключение

Внедрение и эффективное использование стандартов на программные интерфейсы связаны с большим количеством сложных проблем, как технического характера, так и экономических и социальных. Тем не менее, все эксперты сходятся на том, что эта деятельность необходима для стабильного развития производства программного обеспечения. Формализация стандартов уже неоднократно предлагалась в качестве возможного решения множества технических задач, связанных с ней.

Однако формализация требует огромных усилий, что позволяет усомниться в применимости подходов на ее основе к реально используемым программным стандартам, достаточно объемным и сложным. Устранить эти сомнения помогут только практические примеры применения подобных методов к таким стандартам. Упомянутый выше проект [15] по формализации LSB Core похоже, является одной из первых попыток формализации значительной части используемого на практике стандарта высокой сложности и позволяет почувствовать все ее выгоды и недостатки.

Мы считаем, что представленный в этой статье подход позволит успешно справляться с подобными задачами. Аргументами в пользу этого мнения являются стабильное продвижение описанного проекта, история применений технологических составляющих данного подхода на практике ([12,14]) и лежащие в его основе базовые принципы хорошего проектирования больших систем [12,13]. Тот факт, что многие инженерные и организационные вопросы также находят отражение в этом подходе, позволяет надеяться, что, избежав участи многих проектов по использованию передовых методик разработки ПО на практике, мы сможем получить действительно полезные практические результаты,.

Для стимулирования деятельности в направлении продвижения открытых стандартов и вовлечения в него широких кругов разработчиков ПО ИСП РАН передает результаты этого проекта и все инструменты, необходимые для работы с ними, сообществу разработчиков ПО с открытым кодом (open source community). Мы надеемся на помощь членов этого сообщества в решении более масштабных задач, таких, как формализация стандартов семейства Linux Carrier Grade, стандартов, регламентирующих встроенные операционные системы и системы реального времени, а также стандартов на библиотеки времени выполнения для широко используемых языков программирования.

Благодарности. Мы благодарим Федеральное агентство по науке и инновациям Российской Федерации (Роснауку) за предоставленную финансовую поддержку для создания при ИСП РАН Центра верификации ОС Linux и проведения проекта по формализации и разработке открытого тестового набора LSB Core.

Литература

[1] *ISO 9646. Information Theory — Open System Interconnection — Conformance Testing Methodology and Framework.* ISO, Geneve, 1991.

[2] *ITU-T. Recommendation Z.500. Framework on formal methods in conformance testing.* International Telecommunications Union, Geneve, Switzerland, 1997.

[3] A. Grinevich, A. Khoroshilov, V. Kuliamin, D. Markovtsev, A. Petrenko, V. Rubanov. *Formal Methods in Industrial Software Standards Enforcement*. Proceedings of the PSI'06 international conference, June 2006.

[4] А. Гриневич, В. Кулямин, Д. Марковцев, А. Петренко, В. Рубанов, А. Хорошилов. *Использование формальных методов для обеспечения соблюдения программных стандартов*. Труды ИСП РАН, том.10, 2006.

[5] http://www.freestandards.org/spec/
[6] http://www.unix.org/version3/ieee_std.html
[7] G. Bernot. *Testing against Formal Specifications: A Theoretical View.* In Proc. of TAPSOFT'91, Vol. 2. S. Abramsky and T. S. E. Maibaum, eds. LNCS 494, pp. 99–119, Springer-Verlag, 1991.
[8] E. Brinksma, R. Alderden, R. Langerak, J. van de Lagemaat, and J. Tretmans. *A formal approach to conformance testing.* In J. de Meer, L. Mackert, and W. Effelsberg, eds. 2-nd Int. Workshop on Protocol Test Systems, pp. 349–363. North-Holland, 1990.
[9] J. Tretmans. *A Formal Approach to Conformance Testing.* PhD thesis, University of Twente, Enschede, The Netherlands, 1992.
[10] I. Bourdonov, A. Kossatchev, V. Kuliamin, and A. Petrenko. *UniTesK Test Suite Architecture.* In Proc. of FME 2002. LNCS 2391, pp. 77–88, Springer-Verlag, 2002.
[11] V. Kuliamin, A. Petrenko, N. Pakoulin, A. Kossatchev, and I. Bourdonov. *Integration of Functional and Timed Testing of Real-time and Concurrent Systems.* In Proc. of PSI 2003, LNCS 2890, pp. 450–461, Springer-Verlag, 2003.
[12] В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. *Подход UniTesK к разработке тестов.* Программирование, 29(6):25–43, 2003.
[13] V. Kuliamin. *Model Based Testing of Large-scale Software: How Can Simple Models Help to Test Complex System.* In Proc. of 1-st International Symposium on Leveraging Applications of Formal Methods, Cyprus, October 2004, pp. 311–316.
[14] V. Kuliamin, A. Petrenko, and N. Pakoulin. *Practical Approach to Specification and Conformance Testing of Distributed Network Applications.* In M. Malek, E. Nett, N. Suri, eds. Service Availability. LNCS 3694, pp. 68–83, Springer-Verlag, 2005.
[15] http://www.linuxtesting.ru
[16] http://www.opengroup.org/bookstore/catalog/c610.htm
[17] http://www.caldera.com/developers/devspecs/
[18] *ISO/IEC 9899. Programming Languages — C.* ISO, Geneve, 1999.
[19] ftp://ftp.gnu.org/gnu/glibc
[20] http://ltp.sourceforge.net
[21] http://www.freestandards.org/en/Download#LSB_Runtime_Testkit
[22] J. F. Leathrum and K. A. Liburdy. *A Formal Approach to Requirements Based Testing in Open Systems Standards.* In Proc. of 2-d International Conference on Requirements Engineering, 1996, pp. 94–100.
[23] E. Farchi, A. Hartman, and S. S. Pinter. *Using a model-based test generator to test for standard conformance.* IBM Systems Journal, 41:89–110, 2002.
[24] http://www.opengroup.org/testing/testsuites/TestSuiteIndex.html
[25] http://posixtest.sourceforge.net/
[26] http://www.osdl.org/lab_activities/carrier_grade_linux
[27] http://www.unitesk.com