Хорошилов Алексей Владимирович
Россия, Москва, Институт системного программирования РАН, khoroshilov@ispras.ru

ФОРМАЛЬНОЕ ОПИСАНИЕ ИНТЕРФЕЙСОВ КАК СРЕДСТВО УЛУЧШЕНИЯ ИНТЕРОПЕРАБЕЛЬНОСТИ

Одной из особенностей современного этапа развития информационных технологий является все возрастающее влияние принципов построения открытых систем, согласно которым важная роль отводится таким атрибутам программного обеспечения как интероперабельность и переносимость. Оба этих понятия основываются на способности разрабатываемой системы взаимодействовать с внешним окружением, которая, как правило, достигается за счет четкого определения интерфейсов взаимодействия. В данной работе мы рассмотрим метод формального описания интерфейсов взаимодействия и покажем, как его применение способствует достижению интероперабельности и переносимости программных систем.

Основным средством достижения переносимости программных систем является следование стандартам на взаимодействие между прикладными программами и системным окружением времени исполнения, таким как стандарт языка Си[1], стандарт на базовые библиотеки операционных систем IEEE 1003.1 (POSIX) [2] или стандарт на базовый бинарный интерфейс операционной системы Linux – ISO/IEC 23360 Linux Standard Base[3]. Принцип работы этого подхода следующий – если программная система следует стандарту на взаимодействие с системным окружением, то она будет переносима между разными программно-аппаратными платформами, предоставляющими, описанное стандартом окружение времени исполнения.

Данный подход успешно работает уже долгое время, несмотря на то, что ряд проблемных вопросов остается открытыми:

- Как удостовериться, что программная система соответствует требованиям стандарта?
- Как удостовериться, что системное окружение соответствует требованиям стандарта?
- А действительно ли программная система, удовлетворяющая требованиям стандарта, будет успешно работать на любой платформе, удовлетворяющей требованиям стандарта?

Рассмотрим, какие методы используются для разрешения поставленных вопросов. Проверка соответствия стандарту программных систем обычно базируется на компиляции исходного кода системы в специальном окружении, которое не позволяет использовать нестандартные возможности. Такая проверка является исключительно статической и не может выявить обращения к стандартным интерфейсам со значениями параметров, поведение системного окружения на которых стандарт не определяет.

Проверка системного окружения на соответствие стандарту обычно проводится более тщательно, так как включает в себя динамическое тестирование специфицируемой функциональности, а не только статические проверки. И хотя наиболее систематичные подходы [4] включают в себя средства аккуратного прослеживания тестируемых требований стандарта, к сожалению, их рекомендациям следуют далеко не всегда.

Для разрешения третьего вопроса наиболее действенным методом является привлечение группы экспертов в соответствующей области для вычитывания стандартов и обнаружения проблемных мест. Этот способ, безусловно, является необходимым, но также необходимы и более технологичные методы обеспечения качества текста стандартов.

Что такое формальное описание интерфейсов взаимодействия и как оно может помочь в данной ситуации? Традиционно стандарты записываются на естественном языке. Формальное, или, другими словами, машинно-читаемое описание, используется в основном только для представления статической семантики отдельных объектов, описываемых в стандарте. Например, для описания форматов данных или типизации интерфейсов взаимодействия.

Заметим, что именно в этих случаях, когда требования к участникам взаимодействий представлены формально, проверка выполнения этих требований вызывает меньше всего проблем.

А что будет, если формальным образом описать не только статические, но и динамические требования к участникам взаимодействий? Если принять во внимание, что исходный код приложений также является формальным описанием, то можно представить себе идеальную картину, на которой специальные математические подходы и инструментальные средства позволяют статически проверить соответствие приложений формализованным требованиям. Но, к сожалению, в настоящее время таких методов верификации, которые справились бы с подобной задачей, не предвидится.

Тем не менее, если наблюдать за происходящими взаимодействиями на формально описанном интерфейсе, то существующие технологии позволяют использовать формальные описания для автоматической проверки соответствия поведения участников взаимодействий требованиям стандарта и сигнализации об обнаружении нарушений. В результате такой подход приводит к шагу вперед в тестировании программных систем на соответствие стандарту, так как позволят организовать более тщательные динамические проверки, которые могут отследить такие проблемы как, обращения к функциям с недопустимыми значениями параметров. Заметим, что в случае динамических проверок недопустимость значений параметров может определяться достаточно сложным образом, с учетом предыдущих взаимодействий и даже с учетом параллельных асинхронных взаимодействий.

Тот же самый подход можно использовать и при тестировании системного окружения на соответствие требованиям стандарта. В простейшем случае, возможность автоматически выносить вердикт о корректности позволяет при разработке тестов сконцентрироваться только на подготовке тестовых данных и организации тестовых ситуаций. Но кроме того, эта возможность открывает путь для применения техник автоматической генерации тестов. Конечно, тесты можно генерировать и без этого, но ценность тестов без каких-либо проверок будет невелика. Также следует отметить, что существуют техники генерации тестовых воздействий [5,6], которые могут использовать формальное описание интерфейса для получения более нацеленных тестов. Таким образом, наличие формального описания интерфейса создает предпосылки для создания тестового набора, обеспечивающего более тщательное тестирование системного окружения, чем при разработке тестов без него.

Итак, формальные описания интерфейсов обещают возможность достичь более качественного уровня проверки соответствия стандарту участников стандартизируемых взаимодействий. А могут ли они автоматически разрешить проблему адекватности стандарта поставленных задачам, которая была сформулирована в третьем вопросе? В общем случае, нет, так как для этого потребуется формально описать не только интерфейс взаимодействия, но и детали его реализации. В качестве иллюстрации данного утверждения, рассмотрим одну из проблем адекватности, обнаруженных в стандарте LSB [3].

Стандарт LSB определяет базовый бинарный интерфейс операционной системы Linux, перечисляя набор системных библиотек, которые должны присутствовать во всех LSB-совместимых системных окружениях и список функций, экспортируемых каждой из этих библиотек. При этом стандарт не накладывает никаких ограничений на экспортирование нестандартных функций из стандартных библиотек. С другой стороны, LSB требует от LSB-совместимых приложений, чтобы их исполнимые файлы зависели только от перечисленные в стандарте системных библиотек и импортировали из них только перечисленные в стандарте функции. И так как для каждой конкретной функции контракт между приложением и системным окружением в стандарте LSB четко определен, предполагается, что любое LSB-совместимое приложение будет успешно работать в любом LSB-совместимом системном окружении.

Однако, стандарт LSB и поддерживающие его инструменты тестирования соответствия не учитывают исторически сложившуюся особенность разделяемых библиотек в системах UNIX, которая заключается в том, что имена экспортируемых и импортируемых функций никак не привязаны к именам содержащих их библиотек. То есть приложение перечисляет список

необходимых ему внешних функций и перечисляет список библиотек, в которых необходимо эти функции искать, а загрузчик приложения выбирает первую попавшуюся функцию с требуемым именем из системных библиотек, перечисленных во втором списке. В результате, может сложится ситуация, когда приложение будет требовать только предписанные стандартом функции из только предписанных стандартом библиотек, но это LSB-совместимое приложение будет некорректно работать на некоторых LSB-совместимых системных окружениях, в которых одна из необходимых стандартных функций будет присутствовать в нескольких стандартных библиотеках, причем в той библиотеке, которая предписана функции стандартом, она будет работать как ей положено по стандарту, а в другой стандартной библиотеке – функция с тем же именем будет работать как-то иначе.

Данный пример демонстрирует, что имея только формальное описание интерфейса взаимодействия, без учета деталей реализации этого интерфейса, невозможно формальными методами выявить все проблемы адекватности стандарта. Но, это не означает, что формальные методы не могут оказаться полезными для обнаружения внутренних проблем формализованного стандарта, таких противоречивость или неоднозначность. Существует множество подходов по валидации формальных моделей, которые позволяют выполнить такого рода проверки.

И все-таки, главным достоинством подхода, связанного с формальным описанием интерфейсов, является выявление множества проблем в тексте стандарта, которое происходит в процессе его формализации. Это связано, в первую очередь, с систематичностью деятельности по формализации требований стандарта, включающей в себя аккуратное чтение текста стандарта, понимания его смысла и преобразование в формальное представление, стимулирующее однозначность, полноту и непротиворечивость формулировки требований. Адекватность стандарта поставленным задачам является наиболее сложным для выявления видом проблем, так как требует учета свойств среды, в которой происходят взаимодействия и других внешних факторов. Тем не менее, при использовании формальной модели для создания инструментария, проверяющего соблюдение контракта между участниками взаимодействия, происходит состыковка формальных моделей с реальной средой и реальным окружением, в результате чего выявляются многие проблемы с адекватностью стандарта.

Таким образом, хотя формальное описание интерфейсов не гарантируют выявление всех проблем в стандарте, оно предоставляет достаточно технологичный способ повышения качества текста стандарта. Причем этот подход полезен не только для проверки адекватности, но и для обеспечения внутренних характеристик качества стандарта:

- однозначности, то есть одинаковости понимания формулировок требований экспертами в соответствующей предметной области;
- полноты, то есть охваченности в описании требований всех аспектов и ситуаций, возможных в рамках описанного контекста работы системы;
- непротиворечивости, то есть отсутствия противоречий и расхождений между различными частями текста стандарта;
- минимальности, то есть отсутствия неоправданных дублирований требований.

До сих пор все наши рассуждения происходили вокруг проблемы обеспечения переносимости программных систем. Аналогичная картина возникает и при рассмотрении проблемы обеспечения интероперабельности, в которой так же ключевую роль играет описание интерфейса взаимодействия между двумя или более сторонами. Причем при рассмотрении динамических аспектов интероперабельности, в большинстве случаев, также можно выделить участников взаимодействий, играющих роль поставщиков услуг, и участников, играющих роль потребителей услуг. Только эти роли не закреплены статически, а изменяются в зависимости от конкретного взаимодействия. Тем не менее, формальное описание контракта между участниками взаимодействий может быть выполнено теми же средствами. Поэтому все сделанные выводы при рассмотрении проблемы переносимости относятся в равной степени и к проблеме интероперабельности.

Мы рассмотрели теоретические достоинства метода формализации интерфейсов, ни сказав ни слова его недостатках. Это не означает, что таковых не существует. Основными

недостатками подхода является достаточно высокая трудоемкость, сложность, и как следствие, достаточно высокая стоимость. Но, если теоретические достоинства достигаются на практике, то высокая стоимость оправдывается высоким качеством результата, которого сложно достичь технологичным образом при помощи альтернативных методов.

На практике существует множество различных подходов, позволяющих записать требования стандарта формальным образом. Среди них выделяются следующие виды формальных спецификаций:

- операционные;
- контрактные;
- логические;
- аксиоматические.

Операционные спецификации (также их называют явными или исполнимыми) предоставляют бóльший простор для формальных методов анализа разработанных описаний, но область их применимости ограничивается относительно небольшими моделями. Другие три вида спецификаций, которые часто называют неявными спецификациями, достаточно близки по своим возможностям и отличаются, в основном, математическими концепциями, лежащими в их основе. Аксиоматические спецификации наименее удобны для проверки корректности поведения участников описываемых взаимодействий, поэтому редко применяются на практике. Логические спецификации удобны для описания темпоральных свойств, а контрактные – наиболее легко воспринимаются в среде разработчиков программ. Кроме того, контрактные спецификации по своему духу наиболее подходят для формализация интерфейсов, так как интерфейс и представляет собой контракт между участниками взаимодействий через этот интерфейс.

Вид спецификаций для описания интерфейса определяет теоретические возможности по использованию полученного результата. С практической точки зрения, при выборе формализма, в первую очередь, следует учитывать особенности его инструментальной поддержки и сложность работы с ним, так как именно возможности инструментов будут определять потенциал использования полученных формальных описаний, а сложность работы и обучения – существенную часть стоимости проекта.

Автору не известны процессы формализации интерфейсов, специально предназначенные для валидации исходных документов и автоматизации тестирования на соответствие. Обычно процессы разработки формальных моделей ставят перед собой более локальные задачи – проведение отдельных видов верификации построенных моделей или автоматизации разработки тестовых наборов. Наиболее близким к решению обоих поставленных задач является процесс FOREST [7]. Хотя этот процесс в первую очередь нацелен на построение формальной спецификации требований и разработку на ее основе тестового набора для тестирования соответствия этим требованиям, также существенное внимание в нем уделяется и вопросам обеспечения качества исходного текста стандарта.

Процесс FOREST организован итеративно и состоит из трех основных этапов, соответствующих трем уровням детализации требований, при этом на каждый этап содержит некоторый замкнутый набор работ с требованиями. Трудоемкость выполнения этапов возрастает от первого к третьему. Результаты каждого этапа нужны для выполнения последующих, но могут быть использованы и отдельно. Поэтому при необходимости работа по процессу FOREST может быть остановлена после выполнения любого из этапов, если полученные результаты уже обладают приемлемыми качествами, а работы следующих этапов оказываются недостаточно эффективными в рамках конкретного проекта.

На первом этапе требования извлекаются из исходного текста стандарта и систематизируются. При этом происходит структуризация доступного знания о предметной области. В результате получается каталог требований, в котором требования сформулированы максимально однозначно, с каждым требованием сопоставлен уникальный идентификатор, требования классифицированы, и, возможно, установлены связи между отдельными требованиями. Каталог требований используется на последующих этапах. В соответствии с решаемыми задачами этот этап называется «Составление каталога требований».

Второй этап ориентирован на анализ требований. В качестве основного средства анализа используются концептуальные модели требований. Соответственно, этот уровень называется «Построение концептуальной модели». В ходе разработки и анализа концептуальных моделей происходит реструктуризация знания о предметной области, и производится проверка свойств требований, таких как адекватность, полнота, и т.п. Разработка модели опирается на каталог требований, построенный в процессе первого уровня.

Третий этап нацелен на представление требований в формальном виде. Этот уровень называется «Построение формальной модели». Требования из каталога записываются с использованием математического формализма контрактных спецификаций. Перевод требований из текстового представления в формальное облегчается тем, что знания о предметной области, были реструктурированы и проанализированы на первых двух этапах.

Процесс FOREST применялся на практике во многих проектах:

- в проекте по разработке тестовых наборов для сетевого протокола IPv6;
- при разработке прототипа тестового набора для протокола контроля прав доступа к мультимедийной информации IPMP-2 [8];
- при разработке тестового набора для части базовой библиотеки Java;
- в проекте по разработке набора тестов для проверки соответствия базовому стандарту Linux (Linux Standard Base, LSB [3]).

В результате этих проектов было обнаружено множество проблем в анализируемых стандартах, в том числе и проблем, мешающих достижению переносимости и интер-операбельности. Даже такие отлаженные итеративные процессы разработки стандартов, какие работают в IETF и Austin Group, не помешали обнаружить существенные проблемы в стандартах IPv6 Mobile и POSIX, соответственно.

При этом следует отметить следующую закономерность. Если стандарт привлекает много внимания сообщества и, соответственно, экспертов предметной области, то высокое качество стандарта обеспечивается существующими средствами – при помощи итеративного процесса анализа и обсуждения текста экспертами. Если же стандарт не находится в центре всеобщего внимания, то формализация требований стандарта позволяет существенно поднять его качество.

В подтверждении этого, говорит тот факт, что в стандарте IPv6 проблем с интероперабельностью в наших проектах обнаружено не было. Но это не означает, что проблем не было обнаружено вовсе. Благодаря систематическому процессу формализации было выявлено, что спецификация IPv6 RFC 2460 в отдельных местах неполна. Например, она перечисляет несколько ситуаций, которые должны рассматриваться как ошибки, и несколько ситуаций, которые ошибками не являются. Но этот перечень не содержит ряд важных случаев, в особенности проблемных ситуаций, которые могут возникнуть при сборке пакета из нескольких фрагментов.

Родственный стандарт IPv6 Mobile, который разрабатывался в более узком кругу экспертов, уже содержал несколько проблем с интероперабельностью. Не избежал таких проблем и стандарт POSIX [9] больше всего было проблем зафиксировано в активно развивающихся стандартах IPMP-2 [8] и LSB [9].

Несмотря на то, что процесс FOREST допускает использование различных средств формализации требований, в ходе всех перечисленных выше проектов применялись контрактные спецификации, записываемые на специализированных спецификационных расширениях языков программирования Си и Java. Такой выбор был обусловлен двумя причинами. Во-первых, использование расширений широкораспространенных языков программирования снимало необходимость обучения разработчиков незнакомым языкам спецификаций. Во-вторых, существующая инструментальная поддержка языков позволяла полноценно реализовать потенциал контрактных спецификаций, так как набор инструментов UniTESK [10] поддерживал и автоматическую проверку наблюдаемого поведения участников взаимодействий на соответствие формализованным требованиям стандарта, и нацеленную генерацию тестовых последовательностей, учитывающую построенные формальные модели требований.

В результате, утверждение о потенциально более тщательном уровне тестирования, достигаемом при использовании формального описания интерфейсов и особенно методов нацеленной генерации тестов, также подтверждается практическими данными. Это показывают как измерения покрытия исходного кода тестируемых реализаций, так и большее число обнаруживаемых ошибок. Сравнение по покрытию исходного кода проводилось между тестовым набором, предназначенным для сертификации дистрибутивов Linux на соответствия стандарту LSB 3.1, и тестовым набором OLVER [9], разработанным на основе формальных спецификаций интерфейсов этого стандарта. В результате покрытие кода glibc OLVER'ом превышает более чем на 30%. Также тестовый набор OLVER обнаруживает более 50 несоответствий между поведением реализации и требованиями стандарта, которые пропускаются официальным тестовым набором. Еще одним примером является выявление при помощи тестов, построенных на основе формальных спецификаций, критичной ошибки в реализации Microsoft Research IPv6, которая не обнаруживается наиболее известным тестовым набором для IPv6 – TAHI [11], который используется для проведения сертификации по программе IPv6 Logo.

А вот утверждение о дополнительном уровне качества тестирования программных систем на соответствие требованиям стандарта, которое достигается за счет применения динамических проверок, основанных на формальных спецификациях, пока реальными результатами не подтверждается. Работы по реализации данного подхода находятся в стадии экспериментов. Тем не менее, принципиальных трудностей, как минимум для большинства случаев, не предвидится.

Таким образом, опыт полученный в Институте системного программирования РАН подтверждает, что применение методов формальной спецификации интерфейсов, описанных в стандартах, в действительности, предоставляет технологичный метод повышения качества самих стандартов и поддерживающих их тестовых наборов, способствуя тем самым достижению интер-операбельности и переносимости программных систем. Но более высокое качество результата, достигаемое по сравнению с применением итеративных методов вычитывания текста стандарта экспертами и ручной разработки тестов, достигается за счет бóльших трудозатрат. Поэтому данный метод может быть рекомендован в случаях,

- когда существует потребность в разработке надежного стандарта с хорошей инструментальной поддержкой;
- нет возможности привлечения широкого круга экспертов;
- но есть возможность привлечения коллектива грамотных специалистов.

Также, возможно рекомендовать применение процесса FOREST в некоторых промежуточных условиях, так как его организация позволяет получать на промежуточных этапах работы результаты промежуточного уровня качества.

[1] ISO/IEC 9899-1999. Programming Languages — C. Geneve: ISO, 1999.
[2] IEEE 1003.1-2004. Information Technology — Portable Operating System Interface (POSIX). New York: IEEE, 2004.
[3] ISO/IEC 23360-1-8:2005, Linux Standard Base (LSB) Core Specification 3.1. Geneve: ISO, 2005.
[4] ISO/IEC 13210:1996, Information Technology – Requirements and Guidelines for Test Methods Specifications and Test Method Implementation for Measuring Conformance to POSIX Standard.
[5] Bourdonov, I.B., Kossatchev, A.S., Kuliamin, V.V., Petrenko, A.K. UniTesK Test Suite Architecture. Proc. of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002.
[6] Veanes, M., Campbell, C., Schulte, W., Tillmann, N. Online testing with model programs. In Proceedings of the 10th European Software Engineering Conference. ACM Press, New York, 2005.
[7] Кулямин, В.В., Пакулин, Н.В., Петренко, О.Л., Сортов, А.А., Хорошилов, А.В. Формализация требований на практике. Препринт 13, ИСП РАН, Москва, 2006
[8] Pakoulin, N., Omelchenko, V., Koptelov, A., Petrenko, A., Kossatchev, A., Cheng, C.. MPEG-2 IPMP Conformance Test Suite Development. AVS M1263: 2004-06.
[9] http://www.linuxtesting.org
[10] http://www.unitesk.com
[11] http://www.tahi.org