DOI: 10.15514/ISPRAS–2020–32(6)–1

Мониторинг и тестирование на основе многоуровневых спецификаций программ

[1,2,3] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1] *Д.В. Ефремов, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1,2] *Е.В. Корныхин, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *В.В. Кулямин, ORCID: 0000-0003-3439-9534 <kulyamin@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[1] *И.В. Щепетков, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[3] *НИУ Высшая школа экономики,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20
[4] *Московский физико-технический институт,*
141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. В исследованиях по формальным методам разработки и верификации программ много внимания уделяется вопросам построения многоуровневых спецификаций, отвечающих потребностям методологии пошаговой детализации и итеративной разработки. При верификации программ или их моделей наличие нескольких уровней спецификаций также упрощает доказательство свойств, поскольку, как правило, при добавлении нового уточняющего уровня удается переиспользовать те доказательства, которые были выполнены для более абстрактных уровней модели. При тестировании на основе формальных моделей и при мониторинге систем с целью проверки соответствия поведения системы требованиям, заданными формальной моделью, желательно пользоваться теми же моделями, которые использовались при формальной верификации. На практике такие модели в крупных программных системах являются многоуровневыми, однако опыта их использования как основы тестирования и мониторинга пока не было. В статье рассматриваются различные методы построения многоуровневых моделей, новые возможности, которые удается извлечь за счет комбинации функциональных спецификаций и спецификаций архитектуры реализации, ограничения, которые приходится учитывать при организации тестирования и мониторинга на основе многоуровневых моделей.

Ключевые слова: формальные модели программ; уточнение; модель архитектуры программы

Для цитирования: Петренко А.К., Ефремов Д.В., Корныхин Е.В., Кулямин В.В., Хорошилов А.В., Щепетков И.В. Мониторинг и тестирование на основе многоуровневых спецификаций программ. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 7-18. DOI: 10.15514/ISPRAS–2020–32(6)–1

Благодарности. Работа поддержана грантом РФФИ № 20-01-00568.

Monitoring and Testing Based on Multi-Level Program Specifications

[1,2,3] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1] *D.V. Efremov, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1,2] *E.V. Kornykhin, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *V.V. Kuliamin, ORCID: 0000-0003-3439-9534 <kulyamin@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[1] *I.V. Shchepetkov, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia,
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[3] *National Research University, Higher School of Economics*
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia
[4] *Moscow Institute of Physics and Technology (MIPT)*
141700, Russia, Moscow region, Dolgoprudny, Campus per., 9

Abstract. Research on formal methods of software development and verification focuses on building specifications using incremental and iterative development methodologies. The presence of several levels of specifications simplifies proving of properties, since it is possible to reuse the proofs that were performed for more abstract layers of the model. It is desirable to use the same models that were used for formal verification also in testing of real systems for compliance with the requirements set by these models. In practice, large software systems are described by multi-level models. There was no experience of using such models as the basis for testing and monitoring. The paper discusses various methods for developing multi-level models, new opportunities that can be obtained through a combination of functional specifications and implementation-level refinements, limitations that must be considered during testing and monitoring of real systems for compliance with multi-level models.

Keywords: software formal models; refinement; software architecture models

For citation: Petrenko A.K., Efremov D.V., Kornykhin E.V., Kuliamin V.V., Khoroshilov A.V., Shchepetkov I.V. Monitoring and testing based on multi-level program specifications. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 7-18 (in Russian). DOI: 10.15514/ISPRAS–2020–32(6)–1.

Acknowledgments. This work was supported by the RFBR grant No. 20-01-00568.

1. Введение. Источники потребности в многоуровневых моделях

В последние годы все большее внимание уделяется методической и инструментальной поддержке жизненного цикла разработки доверенного программного обеспечения. В настоящее время имеющаяся цепочка покрывает значительное число процессов разработки и верификации ПО, но есть еще и слабо поддержанные виды работ. Одним из таких видов является тестирование на соответствие требованиям к системе. Причиной такого отставания является то, что в отличие от статического анализа или динамического фаззинга значимый эффект от тестирования на соответствие требованиям возможен только тогда, когда эти требования описаны достаточно полно и формализованы. Хотя инструменты генерации тестов на основе формальных спецификаций или формальных моделей начали появляться примерно 20 лет назад (KVEST 1997 [1], UniTESK 2002 [2], SpecExporer 2004 [3]), они не получили широкого распространения из-за значительной трудоемкости разработки формальных спецификаций, необходимых для генерации тестов.

Для разработки собственно спецификаций или моделей программ, из которых можно было бы генерировать тесты, были потрачены значительные усилия. Наиболее известными технологиями разработки спецификаций или моделей, из которых можно генерировать тесты, являются технологии SDL и UML, языки для описания тестов (или абстрактных тестов), например, TTCN-3, спецификационные расширения языков программирования JML

(Java Modeling Language [4]), Spec# [5], ACSL (ANCI/ISO C Specification Language [6]), SPARK (SPARK Ada) [7].

Сейчас ситуация в области разработки ответственного программного обеспечения меняется, вводятся новые стандарты и регламенты, которые используются при сертификации программных систем, где требования надежности, защищенности и информационной безопасности имеют первоочередное значение. Как следствие, у разработчиков доверенного ПО появляются сначала достаточно подробные спецификации интерфейсов программных систем (как на системном уровне, так и на уровне подсистем и модулей), а потом и формальные спецификации. Это позволяет по-новому взглянуть на задачи тестирования на основе формальных спецификаций или формальных моделей, так как основной барьер, мешающий распространению тестирования на основе формальных спецификация и/или моделей, постепенно преодолевается.

Уже первые проекты по внедрению технологий поддержки жизненного цикла доверенного ПО на примере верификации средств защиты операционной системы Astra Linux Special Edition [8] показали, что модели системы в реальной практике более сложны и разнообразны по сравнению с теми, которые рассматривались в первые годы появления технологий тестирования на основе моделей. Первой задачей, которая встала перед разработчиками методов и инструментов генерации тестов из формальных спецификаций в последние годы, стала задача установления отношения между требованиями к системе в целом (внешним интерфейсам) и требованиями к отдельным модулям (внутренним интерфейсам). Вторая задача состояла в том, что набор понятий, при помощи которого описываются некоторые важные свойства системы (например, свойства информационной безопасности), не совпадает и непросто отображается на понятия, в которых описываются интерфейсы реализации (в конкретном случае внешние и внутренние интерфейсы операционной системы).

Обе задачи возникают из-за того, что в большом программном проекте мы вынуждены работать с так называемыми «многоуровневыми» спецификациями. Далее в статье будут рассмотрены причины появления таких спецификаций, виды «многоуровневости» и выводы, которые приходится делать при адаптации известных схем генерации тестов и мониторинга в данном контексте.

Основными источниками введения многоуровневости спецификаций в программной инженерии являются следующие:

1) **Методология программирования «сверху-вниз»**, пошаговая детализация. Отправная точка этих методологий – инженерный опыт в технических отраслях, который сводится к выделению нескольких стадий конструирования/производства нового изделия: замысел-эскиз-макет/прототип-первая рабочая версия и так далее. Практическое преимущество такого многостадийного процесса состоит в том, что он позволяет разумно тратить усилия конструкторов, накапливать опыт и находить проблемы на ранних стадиях, экспериментировать тогда, когда эксперименты еще не слишком затратны как по средствам, так и по времени.
2) **Методы аналитической верификации**. Эти методы на практике столкнулись с чрезвычайно высокой «стоимостью» ошибок в спецификациях, в частности, из-за неправильного понимания требований к функциональности нового разрабатываемого продукта: при обнаружении таких дефектов переделка спецификации и последующая «переверификация» требовали огромных (к сожалению, часто многократно возрастающих) затрачиваемых усилий. Кроме того, ограничения инструментов верификации подталкивали строить многоуровневые модели (спецификации). В этом случае при появлении нового «уточняющего» уровня задача верификации оказывается существенно менее сложной, что упрощает работу верификатора и позволяет уложиться в ограничения инструмента верификации.
3) **Задачи анализа защищенности программных систем**. Постановка задач информационной безопасности требует тесного взаимодействия специалистов по

информационной безопасности и программистов. Привычные спецификации интерфейсов, предназначенные для описания функциональности программной системы, нацелены на описание возможностей системы, которые она должна предоставлять пользователю (например, разработчику приложений, использующих специфицируемые интерфейсы). Для специалиста по информационной безопасности главный аспект требований – это правила ограничения предоставления информации. То есть, с одной стороны, это некоторая новая группа требований, а с другой стороны, это дополнительные требования к тем же сущностям, которые были описаны в спецификациях программных интерфейсов. То есть здесь возникает необходимость «приподнять» уровень функциональных требований, чтобы в компактном и ясном виде сформулировать именно требования информационной безопасности.

4) **Задачи распространения требований системного уровня на отдельные внутренние компоненты.** Эта задача встречалась и раньше, в частности, при проектировании и верификации телекоммуникационных протоколов, но в последнее время востребованность в грамотном решении этой задачи выросла в связи с проблемами верификации (и сертификации) систем критических по надежности и безопасности. В силу того, что сами по себе системы, нуждающиеся в тщательном анализе и верификации, могут быть настолько крупными, что имеющиеся методы верификации их целиком проанализировать не позволяют, современные требования регуляторов заключаются в том, что для целевой системы сначала должен быть проведен архитектурный анализ, выделены критические компоненты (см., например, ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования»). Затем критические компоненты подвергаются настолько, насколько это возможно, тщательному анализу/верификации.

   Типичными примерами таких компонентов являются криптографические модули или модули безопасности в операционных системах и СУБД (например, Linux Security Module – LSM, в операционной системе Linux). Проблема спецификации LSM – это типичная проблема построения спецификации интерфейсов внутреннего компонента системы в соответствии с требованиями системного уровня. На системном уровне при этом следует рассматривать как минимум два уровня спецификаций. Самый верхний – это так называемая модель политики безопасности или модель управления доступом (Security Policy Model - SPM), которая сама так же может быть многоуровневой. Нижний уровень – функциональная спецификация внешних интерфейсов системы (Functional SPecification – FSP). Например, в случае ядра операционной системы, FSP – это спецификация системных вызовов.

Тем самым можно заключить, что потребность в разработке многоуровневых спецификаций возникла достаточно давно, но в последние годы в секторах, связанных с задачами обеспечения высокой надежности и защищенности программных систем, потребность уже переросла в критически важную необходимость. Требования разработки таких моделей и многоуровневых спецификаций предписываются рядом стандартов (например, ГОСТ Р ИСО/МЭК 15408-2012 и КТ-178С), где, в частности, речь идет о формальных моделях политик безопасности или о высокоуровневых и низкоуровневых спецификациях интерфейсов.

Традиционно считается, что формальная спецификация поведения системы нужна, в основном, как стартовая точка для формальной верификации системы. В действительности полезность разработки формальной спецификации не ограничивается только поддержкой формальной верификации. Разработка формальных спецификаций сама по себе положительно влияет на качество программного обеспечения, что многократно отмечалось специалистами, использовавшими формальные методы на практике – это объясняется тем, что при разработке формальной спецификации функционирование системы рассматривается в новом ракурсе. Спецификатор думает не столько о том, как можно реализовать ту или иную

функцию, сколько о том, каковы могут быть области допустимых значений входных параметров и критерии корректности реализации.

Тем не менее, весь потенциал формальных методов раскрывается тогда, когда на основе формальной спецификации проводятся различные виды верификации, включая проверку моделей (model checking) и тестирование. Необходимо заметить, что объектом верификации служит как сама модель, так и реализация, которая должна отвечать требованиям, представленными моделью/спецификацией.

Для крупных программных комплексов одним видом верификации, например, аналитической верификацией или проверкой моделей, ограничиваться нельзя в силу того, что современные методы и инструменты не справляются с такой задачей. Поэтому имеет смысл использовать спецификации для генерации тестов – такой подход называется тестированием на основе формальных моделей или на основе формальных спецификаций (Model/Specification Based Testing – MBT и SBT).

Цель данной статьи – проанализировать, какие формы многоуровневых спецификаций встречаются в реальной практике, как они используются и могут использоваться в автоматизации тестирования.

2. Виды многоуровневых моделей

Для более подробного рассмотрения методов построения многоуровневых спецификаций, рассмотрим отдельно спецификации на языках моделирования и на спецификационных расширениях языков программирования.

К первой группе относятся такие языки как ASML [9], B [10], Event-B [11], TLA+ [12], VDM [13], Z [14].

Исторически первым из перечисленных методов появился **VDM (Vienna Development Method** – Венский метод разработки программ). В настоящее время существует три диалекта нотации языка: VDM-SL, позволяющий описывать абстрактные типы данных с пред- и постусловиями функций; VDM++ расширяет VDM-SL возможностями задания объектно-ориентированных и параллельных моделей; VICE, который добавляет к VDM++ средства для описания приложений реального времени и распределенных систем.

В VDM определен ставший классическим подход к уточнению моделей. Это схема 1-to-1, то есть одной структуре данных и одной абстрактной операции должна соответствовать одна детализированная (уточненная) структура данных (тип данных) или одна операция в случае уточнения операций. Однако при этом ограничения на способ описаний уточнения очень незначительные: связь между уточненной сущностью и уточняемой задается произвольной функцией абстрагирования `retr` (от англ. retrieve – вернуться в данном случае к исходному, менее детальному описанию). Функция `retr` определяется для всех уточняемых (impl) типов данных:

```
retr: impl_Type -> abstr_Type
retr_Type_result (impl_Foo (retr__agrument)) =
  abstr_Foo (abstr_agrument)
```

Замечание. Хотя формально ограничение 1-to-1 для соответствия типов данных в VDM имеется, в случае описания соответствия сигнатур функций можно рассматривать кортеж аргументов и кортеж результатов как два (неименованных) типа данных. Это позволяет описывать отношения уточнения более гибко, чем в случае строгой трактовки 1-to-1. Но, в свою очередь, это затрудняет аналитическое доказательство соответствия. Если функция не является «чистой», то есть имеется зависимость от глобальных переменных или функция может иметь побочный эффект, это еще больше затрудняет доказательство соответствия.

На практике VDM применялся для моделирования и верификации, прежде всего, систем реального времени и встроенных систем управления. В последние годы разработчики VDM

сместили основное внимание в область моделирования киберфизических систем. Здесь также стоят задачи разработки и верификации многоуровневых моделей, но эта тематика требует отдельного исследования и не рассматривается в данной статье. Имеется список промышленных и исследовательских проектов, использовавших VDM [15].

Формальный метод B основан на теории множеств и логике предикатов первого порядка. Он поддержан коммерческим инструментом, имеется ряд примеров промышленного применения метода, в частности он использовался для разработки многоуровневой формальной модели операционной системы fmC/OS [16]. Процесс разработки был разбит на две части: предварительно была разработана абстрактная модель верхнего уровня, включающая в себя модели 8 подсистем, а затем и модель нижнего уровня, представляющая собой уточнение абстрактной модели. Далее на основе детальной модели был автоматически сгенерирован исполняемый код операционной системы. В работе утверждается, что каждый уровень модели (абстрактный и детальный) был полезен и позволил обнаружить и исправить ряд ошибок.

Формальный метод B был также использован для моделирования механизма контроля доступом в ядре ОС Linux, который включает в себя описание аппаратной изоляции (ARM TrustZone) [17]. Разработанная формальная модель является многоуровневой и состоит из четырех абстрактных машин, которые описывают процессы операционной системы, ресурсы и политику доступа. Четвертая машина включает в себя три предыдущие для получения доступа к определенным переменным и операциям. Она была частично верифицирована с использованием инструментов автоматического доказательства платформы Atelier B и инструмента проверки моделей ProB.

Примеры показывают, что метод применим и для задания абстрактной модели верхнего уровня и для детальной спецификации, а также для доказательства отношения уточнения между ними. Однако метод предлагает не очень наглядные средства для композиции моделей при помощи конструкций INCLUDES, USES, SEES, в то время как каждый модуль модели описывается в отдельном файле. Отношение уточнения может быть типа 1-to-n, [18].

Метод предполагает генерацию последовательного исходного кода на основе модели в программы на языках C, C++, Java и Ada. Наличие такой возможности является важным достоинством метода, однако в случае системного ПО, этот механизм требует специальных доработок, что при использовании импортного коммерческого продукта весьма проблематично.

Для моделирования реактивных систем широко используется модифицированная нотация, которая получила название Event-B. Она поддерживается открытыми инструментами, также позволяет проводить дедуктивную верификацию, проверку моделей, позволяет строить и верифицировать многоуровневые модели и генерировать программы на языках программирования из моделей, которые детализированы в достаточной степени.

Метод ASM (Abstract State Machines) основан на расширенных конечных автоматах и построении их композиции. Данный метод успешно применялся для моделирования и верификации программных систем, предоставляющих программный интерфейс. Есть работы по применению ASM для генерации тестов компиляторов, распределенных систем и других видов ПО.

Метод ASM предоставляет возможность разработки многоуровневых моделей и пошагового уточнения таких моделей. В работах Э. Бёргера (E. Börger) и Р. Штерка (R. Stärk) рассматривается много видов уточнения, в частности, достаточно мощный механизм для выполнения доказательства отношений уточнения, в том числе по схеме n-to-m. При уточнении в ASM удается показать не только изменение структуры данных в моделях (data refinement), но и изменение в поведении (procedural refinement).

Метод успешно позволяет моделировать параллельные системы. В то же время при помощи расширенных автоматов представляется трудным формулировать глобальные требования и

требования живости (liveness). По этой причине метод не удобен для задания абстрактных моделей верхнего уровня, но предоставляет много средств для определения детальной спецификации.

В целом можно отметить, что Microsoft Research потратило много усилий для разработки инструмента MBT на основе ASM (так называемый ASMT). Вместе с тем, через 2 года экспериментов от нотации ASML пришлось отказаться, так как она оказалась неудобной для программистов-практиков. Кроме того, из публикаций видно, что для тестирования на основе моделей многоуровневые спецификации не применялись.

Инструменты на основе ACSL-спецификаций. В первую очередь это инструменты Frama-C для дедуктивной верификации C-программ. Имеется также расширение E-ACSL – Executable ACSL – Исполнимое подмножество ACSL, которое используется для спецификации проверочных утверждений (assertions) при тестировании и мониторинге C-программ. ACSL не предлагает никакой общей схемы уточнения и построения многоуровневых спецификаций, но в рамках данной работы он интересен тем, что предлагает некоторый единый язык спецификаций, который можно использовать и для аналитической верификации, и для тестирования.

Вывод. На основе рассмотренных работ можно заключить, что специалисты по формальным методам активно используют многоуровневые модели в аналитической верификации, но не пользуются такими моделями при тестировании на основе моделей. Это, по-видимому, объясняется чрезмерным усложнением системы тестирования и отсутствием принципиально нового качества, которого авторы публикаций пока не видят при тестировании на основе многоуровневых моделей.

3. *Использование многоуровневых моделей для тестирования и мониторинга*

Вместе с тем опыт построения тестовой системы для проверки средств защиты информации (СЗИ) операционной системы Linux [8] показал, что потенциал многоуровневых моделей может быть использован в более полной мере. Этот подход, в частности, может быть использован при организации тестирования системного интерфейса операционной системы (системные вызовы – system calls) на соответствие модели управления доступом и при тестировании внутренних модулей системы на проверку соответствия требованиям, сформулированным на более высоких уровнях, например, на уровне той же модели управления доступом или на уровне системных вызовов. Заметим, что роль высокоуровневых моделей при организации тестирования может быть различной. Такие модели могут использоваться для:

- фильтрации (отбраковки) некорректных входных тестовых данных – на основе предусловий;
- генерации тестового оракула (автоматического анализатора полученного результата) - на основе постусловия;
- оценки полученного тестового покрытия (на основе структуры пред- и постусловий);
- построения модели тестовых последовательностей, например, так, как это делается в UniTESK.

Данная работа не ставит своей целью дать детальное описание тестовой системы, работающей с многоуровневыми спецификациями. Здесь мы опишем только общую схему построения такой системы.

На рис. 1 показана достаточно простая схема использования двухуровневой спецификации. Модель в данном случае используется для определения (и автоматической генерации) «тестового оракула» – программного компонента системы тестирования, который в динамике анализирует корректность поведения тестируемой системы. В данном случае оракул проверяет, соответствует ли результат очередного системного вызова спецификации этого

вызова (Оракул Д). В дополнение к Оракулу Д на основе Модели управления доступом можно построить Оракул А. Вместе с тем, даже в такой простой ситуации построение Оракула А может оказаться сложной задачей, а поскольку оракул работает во время тестирования, его вычислительная сложность и потребности в ресурсах памяти может оказаться чрезмерными. По этой причине в промышленном проекте по верификации средств защиты информации операционной системы Astra Linux Special Edition от Оракула А пришлось отказаться – на рисунке этот оракул изображен при помощи пунктирной линии.

Рис. 1. Схема мониторинга и анализа трассы на уровне системных вызовов ОС
Fig. 1. Trace monitoring and analysis scheme at the level of OS system calls

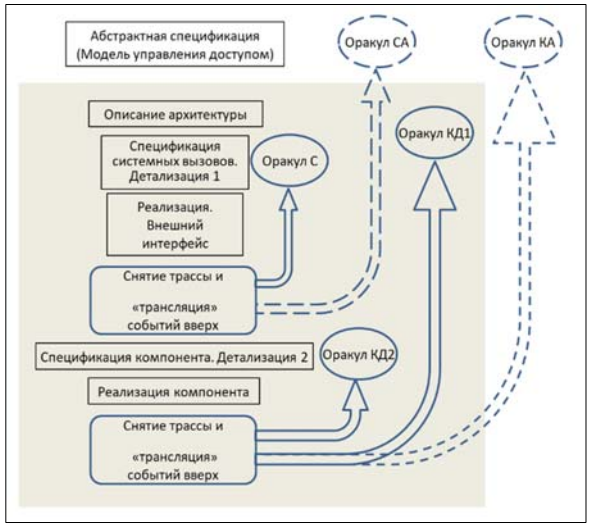

Рис. 2. Схема мониторинга и анализа трассы на уровне системных вызовов и внутреннего компонента ОС
Fig. 2. Trace monitoring and analysis scheme at the level of system calls and internal OS component

Более сложная схема использования многоуровневых моделей показана на рис. 2. Здесь целевыми объектами тестирования является ядро ОС (системные вызовы) и внутренний компонент ядра. В случае защищенной операционной системы семейства Linux, это может быть так называемый модуль безопасности (Linux Security Module – LSM). Функциональность LSM определяется Моделью управления доступом, но является абстрактной по отношению к спецификации интерфейса LSM (то есть не содержит всех деталей, необходимых для спецификации функций LSM. В частности, в модели не описаны сложные структуры данных, с которыми работают программы ядра ОС, а они являются важной составляющей описания поведения LSM.

На рис. 2 также два оракула отмечены пунктирными линиями – эти оракулы реализовать крайне сложно.

Важно отметить, что причиной повышенной сложности построения оракулов в приведенных примерах является сложность отношения уточнения, которая выбирается в том или ином случае, а сложность отношения в свою очередь зависит от степени различия интерфейсов моделей и реализаций. Можно сказать, что всегда, когда выбран (или пришлось выбрать) механизм уточнения типа 1-to-n или n-to-m, оракул оказывается слишком сложным. Опыт ИСП РАН по верификации моделей управления доступом и ядер операционных систем показал, что имеется два способа отказаться от прямолинейного решения динамической проверки на основе многоуровневых спецификаций. Первый способ был использован в проекте по верификации ОС Astra Linux Special Edition [19]. При разработке спецификаций системных вызовов статически было доказано, что выполнение требований функциональной спецификации системных вызовов влечет выполнение требований безопасности, сформулированных в Модели управления доступом. Это означает, что при динамической проверке выполнения системных вызовов (при анализе трассы, где сохраняется информация об аргументах и результатах системных вызовов) положительный вердикт Оракула Д (рис. 1) гарантирует выполнения требований Модели управления доступом. Эта схема была выбрана по той причине, что Модель и спецификация были связаны отношением n-to-m.

При разработке Модели управления доступом для ОС Linux Альт 8 СП было принято решение приблизить структуру интерфейсов Модели к структуре системных вызовов. Это существенно упростило отношение уточнения, по этой причине тестирование в данном случае можно организовать с использованием Оракула А, что не будет являться слишком сложной задачей [20].

При тестировании корректности использования внутренних компонентов системы на соответствие общесистемным требованиям необходимо помимо спецификации внешнего (наблюдаемого извне) поведения системы иметь в распоряжении спецификацию архитектуры реализации и значимых протоколов взаимодействия внутренних компонентов. Так в случае модуля LSM важным требованием безопасности является вызов соответствующей функции LSM всегда, когда операционная система разрешает доступ к некоторому объекту (запретить доступ операционная система иногда может и без обращения к LSM). Такого рода протоколы либо должны включаться в спецификацию архитектуры, либо извлекаться из спецификации архитектуры, желательно, простым способом. На примере, приведенном на рис. 2, показано, что Оракул КД2 определяется на основе спецификации архитектуры и спецификации системных вызовов.

4. Заключение

В статье представлены результаты первого года работ по гранту РФФИ 20-01-00568 по теме «Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы». Были исследованы различные способы построения и верификации многоуровневых формальных моделей (спецификаций) программ и проанализированы проблемы использования таких спецификаций для целей тестирования и мониторинга. На

основе обобщения опыта работ ИСП РАН по верификации средств защиты информации операционных систем и других программных комплексов были сформулированы основные принципы построения тестовой системы на основе многоуровневых спецификаций.

Для апробации предложенного подхода мы разработали экспериментальную систему тестирования на основе многоуровневых спецификаций. В настоящее время удалось решить следующие задачи.

1. разработан метод уточнения на основе состояний для формального метода Event-B. Данный метод позволяет упростить процессы разработки и верификации многоуровневых спецификаций в ситуациях, когда интерфейсы различных уровней спецификации существенно отличаются друг от друга;
2. с использованием предложенного метода разработана многоуровневая спецификация, которая может быть использована для демонстрации возможностей экспериментальной системы тестирования;
3. разработан компонент мониторинга системных вызовов и внутренних событий ядра Linux на основе kprobes/kretprobes. Система включает в себя модуль ядра для мониторинга и приложение-демон пользовательского пространства для сбора информации от модуля ядра;
4. разработаны компоненты для сбора начального состояния операционной системы перед выполнением тестов, для преобразования трасс событий, снятых с ядра Linux, в вид пригодный для воспроизведения на многоуровневой спецификации из пункта 2;
5. на основе ProB разработан инструмент воспроизведения трасс из пункта 4 на многоуровневой спецификации из пункта 2;
6. на основе знаний о системе Linux, многоуровневой формальной спецификации из пункта 2, включающей в себя уровни дискретного контроля доступа (DAC), списков управления доступом (ACL), подсистемы контроля целостности (IMA/EVM), мандатного контроля доступа, разработана методика построения тестового набора для их проверки на реальной системе;
7. в соответствии с методикой построения тестового набора создан тестовый пакет программ, покрывающих разные виды доступа, а также перечисленные в пункте 6 системы контроля доступа;
8. разработанные компоненты из пунктов 2, 3, 4, 5 и 7 объединены в единую схему запуска тестов на системе GNU/Linux, снятия трасс событий с этих тестов, трансляции трасс в вид, пригодный для воспроизведения на многоуровневой спецификации, и, собственно, воспроизведения оттранслированных трасс на многоуровневой формальной спецификации для проверки соответствия реального поведения системы с модельным в рамках экспериментального программного комплекса.

Развитие предложенного подхода планируется направить на пополнение техник мониторинга техниками генерации тестов на основе многоуровневых спецификаций.

Список литературы / References

[1] I. Burdonov, A. Kossatchev, A.K. Petrenko, D. Galter. Kvest: Automated generation of test suites from formal specifications. Lecture Notes in Computer Science, vol. 1708, 1999, pp. 608-621.

[2] I.B. Bourdonov, A.S. Kossatchev, V.V. Kuliamin, A.K. Petrenko. UniTesK test suite architecture, Lecture Notes in Computer Science, vol. 2391, 2002, pp. 77-88.

[3] SpecExporer. URL: https://docs.microsoft.com/ru-ru/archive/msdn-magazine/2013/december/model-based-testing-an-introduction-to-model-based-testing-and-spec-explorer, accessed 21.12.2020.

[4] JML (Java Modeling Language), URL: http://www.eecs.ucf.edu/~leavens/JML//index.shtml, accessed 21.12.2020.

[5] M. Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. Lecture Notes in Computer Science, vol. 3362, 2004, pp. 49-69.

[6] ACSL (ANCI/ISO C Specification Language). https://www.academia.edu/16532644/ACSL_ANSI_ISO_C_Specification_Language, accessed 21.12.2020.
[7] SPARK. http://www.spark-2014.org, accessed 21.12.2020.
[8] П.Н. Девянин, Д.В. Ефремов, В.В. Кулямин, А.К. Петренко, А.В. Хорошилов, И.В. Щепетков. Моделирование и верификация политик безопасности управления доступом в операционных системах. М., Горячая линия – Телеком, 2019 г., 214 стр. / P.N. Devyanin, D.V. Efremov, V.V. Kulyamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov. Modeling and verification of access control security policies in operating systems. M., Hotline – Telecom, 2019, 214 p. (in Russian).
[9] Y. Gurevich. Evolving algebras 1993: Lipari guide. In: Börger E. (ed.) Specification and Validation Methods, pp. 9-36. Oxford University Press, Oxford, 1995.
[10] J.-R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 2005, 815 p.
[11] J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, 612 p.
[12] L. Lamport. Specifying Concurrent Systems with TLA+. In: Broy M., Steinbrüggen R. (eds). Calculational System Design, pp. 183–247. IOS Press, Amsterdam, 2000.
[13] D. Bjorner, C.B. Jones. The Vienna Development Method: The Meta-Language. Lecture Notes in Computer Science, vol. 61, 1978.
[14] J.-R. Abrial. Data Semantics. In Proc. of the IFIP Working Conference on Data Base Management, 1974, pp. 1–59.
[15] C.B. Nielsen, P.G. Larsen, J. Fitzgerald, J. Woodcock. Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Computing Surveys (CSUR), vol. 48, issue 4, 2015, article no. 18.
[16] C. Danmin, S. Yue, C. Zhiguo. A Formal Specification in B of an Operating System. The Open Cybernetics & Systemics Journal, no. 9, 2015, pp. 1124-1129.
[17] L. Ren, R. Chang, Q. Yin, W. Wang. Using the B Method to Formalize Access Control Mechanism with TrustZone Hardware Isolation. Lecture Notes in Computer Science, vol. 10701, 2017, pp. 759-769.
[18] S. Hallerstede. M. Hasangic, S. Krings, P.G. Larsen, M. Leuschel. From Software Specifications to Constraint Programming. Lecture Notes in Computer Science, vol. 10886, 2018, pp. 21-36.
[19] D. Efremov, I. Shchepetkov. Runtime Verification of Linux Kernel Security Module. Lecture Notes in Computer Science, vol. 12233, 2020, pp. 185-199.
[20] V. Kuliamin, A. Khoroshilov, D. Medvedev. Formal Modeling of Multi-Level Security and Integrity Control Implemented with SELinux. In Proc. of the 2019 Actual Problems of Systems and Software Engineering (APSSE), 2019, pp. 131-136.

Информация об авторах / Information about authors

Александр Константинович ПЕТРЕНКО – доктор физико-математических наук, профессор, начальник отдела ИСП РАН, профессор МГУ и ВШЭ. Область интересов: формальные методы программной инженерии, тестирование программного и аппаратного обеспечения, формальная спецификация требований.

Alexander Konstantinovich PETRENKO – Doctor of Physical and Mathematical Sciences, Professor, Head of the Department of ISP RAS, Professor of Moscow State University and Higher School of Economics. Areas of interest: formal methods of software engineering, testing of software and hardware, formal specification of requirements.

Денис Валентинович ЕФРЕМОВ – младший научный сотрудник ИСП РАН. Основные научные интересы: формальная верификация, статический и динамический анализ операционных систем, инфраструктура разработки операционных систем.

Denis Valentinovich EFREMOV – Junior Researcher at ISP RAS. Areas of Interest: formal verification, static and dynamic analysis of operating systems, operating systems development infrastructure.

Евгений Валерьевич КОРНЫХИН – кандидат физико-математических наук, доцент кафедры системного программирования МГУ, старший научный сотрудник ИСП РАН. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Eugeny Valerievich KORNYKHIN – Ph.D. in Physics and Mathematics, Associate Professor of system programming departments at Moscow State University and Senior Researcher at ISP RAS. Fields of Interest: formal deductive verification, model-based testing.

Виктор Вячеславович КУЛЯМИН – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН, доцент кафедр системного программирования МГУ и ВШЭ. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Viktor Vyacheslavovich KULIAMIN – Ph.D. in Physics and Mathematics, leading researcher at ISP RAS, associate professor of system programming departments at Moscow State University and the Higher School of Economics. Fields of Interest: formal deductive model verification, model-based testing

Алексей Владимирович ХОРОШИЛОВ – кандидат физико-математических наук, ведущий научный сотрудник, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV – Ph.D. in Physics and Mathematics, Leading Researcher, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at Moscow State University, the Higher School of Economics, and Moscow Institute of Physics and Technology. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Илья Викторович ЩЕПЕТКОВ – стажер-исследователь. Область интересов: разработка и верификация формальных моделей компьютерных систем

Ilya Viktorovich SHCHEPETKOV – Intern Researcher. Fields of Interest: development and verification of formal models of computer systems