DOI: 10.15514/ISPRAS-2025-37(6)-49

Поддержка процессов верификации средств защиты информации на основе формальных моделей политик управления доступом: инструмент АНИС

[1,2] *А.А. Карнов, ORCID: 0000-0002-2066-9946 <karnov@ispras.ru>*
[1,3] *Е.В. Корныхин, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*

[1] *Институт системного программирования имени В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

[2] *НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20.*

[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.*

Аннотация. В статье исследуются проблемы внедрения технологий верификации программных систем ответственного назначения, в частности, средств защиты информации, на основе формальных методов разработки и верификации программ. Обосновывается выбор метода динамической верификации при верификации средств защиты информации операционных систем общего назначения, где на средства защиты информации возлагается контроль предоставления доступом к информационным объектам, таким как файлы, директории, пользовательские процессы и др. В качестве примера реализации рекомендаций ГОСТ Р 59453.4-2025 «Защита информации. Формальная модель управления доступом. Часть 4. Рекомендации по верификации средства защиты информации, реализующего политики управления доступом, на основе формализованных описаний модели управления доступом» описывается инструмент АНИС. Рассматривается вопрос об ограничения на язык описания формальных моделей управления доступом для его использования в технологии динамической верификации, а также другие вопросы, которые важны для обеспечения требований использования инструмента в практике верификации и сертификации средств защиты информации.

Ключевые слова: Event-B; динамическая верификация; формальная модель; Python.

Для цитирования: Карнов А.А., Корныхин Е.В., Петренко А.К. Поддержка процессов верификации средств защиты информации на основе формальных моделей политик управления доступом: инструмент АНИС. Труды ИСП РАН, том 37, вып. 6, часть 4, 2025 г., стр. 45–60. DOI: 10.15514/ISPRAS–2025–37(6)–49.

Security Protection Verification based on Formal Security Policy Models: ANIS Toolset

[1,2]*A.A. Karnov, ORCID: 0000-0002-2066-9946 <karnov@ispras.ru>*
[1,3]*E.V. Kornykhin, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3]*A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

[2] *National Research University, Higher School of Economics*

20, Myasnitskaya st., Moscow, 101978, Russia.

[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. This paper explores the implementation of verification technologies for mission-critical software systems, in particular information security tools, based on formal methods of program development and verification. It also substantiates the choice of a dynamic verification method for verifying information security tools for general-purpose operating systems, where information security tools are responsible for controlling access to information objects such as files, directories, user processes, etc. The ANIS tool is described as an example of implementing the recommendations of GOST R 59453.4-2025 "Information protection. Recommendations on verification of formal access control model. Part 4. Recommendations for verification of information security features that implement access control policies based on formal descriptions of the access control model". The article also discusses restrictions on the language for describing formal access control models for use in dynamic verification technology, as well as other issues that are important for ensuring the requirements for using the tool in the practice of verifying and certifying information protection means.

Keywords: Event-B; runtime verification; formal model; Python.

For citation: Karnov A.A., Kornykhin E.V., Petrenko A.K. Security Protection Verification based on Formal Security Policy Models: ANIS Toolset. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 6, part 4, 2025. pp. 45-60 (in Russian). DOI: 10.15514/ISPRAS-2025-37(6)-49.

1. Введение. Проблемы строгой верификации СЗИ в промышленных программных системах

Задача повышения защищенности программных систем становится одной из важнейших в современном мире. В особо ответственных системах, таких как операционные системы, СУБД, сетевой инфраструктуре, выделяется механизм, который называется средства защиты информации (СЗИ). Его назначение – обеспечивать контроль за процессами передачи, хранения и обработки информации с целью выполнения требований информационной безопасности.

Верификация механизма СЗИ – первоочередная задача в обеспечении кибербезопасности, для решения которой необходимо применять весь арсенал доступных средств. Строгая верификация предполагает наличие строгих, однозначно понимаемых требований корректности целевой (верифицируемой) программной системы, полный, или, по крайней мере, систематичный перебор проверяемых ситуаций, и множество проверок поведения целевой системы, которые выполняются в строгом соответствии с требованиями корректности.

Для верификации СЗИ на промышленном уровне работа на каждом этапе процесса верификации должна быть автоматизирована хотя бы частично. Автоматический анализ требований и их трансформация в проверки поведения целевой системы означают, что требования должны быть представлены в машиночитаемом виде. Корректность самих требований, в частности, их непротиворечивость, желательно не только проверять, но и

доказывать формально. Для этого требования должны быть представлены в форме формальных моделей политик управления доступом и формальных спецификаций системы. В 2021-2025 годах были разработаны и введены в действие четыре части стандарта ГОСТ Р 59453 «Защита информации. Формальная модель управления доступом» [1-4] (далее будем писать Стандарт). Одна из его целей – это создание теоретической базы для процессов разработки средств защиты информации на основе формальных методов.

Однако на пути организации верификации СЗИ таких сложных программных систем, как операционные системы или СУБД, стоит много технических и методических проблем, что существенно тормозит внедрение строгого подхода в практику. Наиболее важным фактором является отсутствие инструментов, которые поддерживали бы процесс верификации на всех фазах. При этом отсутствие инструментов объясняется не только сложностью их разработки, а отсутствием гибких и эффективных решений интеграции разных парадигм и разных платформ для разработки и верификации формальных моделей, и традиционных парадигм программирования и платформ разработки и анализа программ.

Упомянутый Стандарт дает некоторые общие рекомендации и требования к отдельным процессам и инструментам моделирования и верификации СЗИ, однако в нем нет ответа на многие технические и методические вопросы как в плане пользовательских нотаций и интерфейсов, так и в плане выбора архитектуры системы верификации.

Перечисленные выше вопросы можно разделить на три группы:

- Выбор схемы верификации: модульная, системная или их комбинация;
- Выбор между статическими и динамическими методами верификации или способ их интеграции;
- Определение базовых сценариев работы верификатора, которые должен поддерживать набор инструментов верификации СЗИ.

В данной статье мы планируем ответить на эти вопросы, опираясь на опыт экспериментальной реализации процессов в рамках инструмента АНИС. При разработке инструмента учитывались как рекомендации Стандарта, так и возможность потенциального промышленного использования.

Данная статья входит в серию статей, посвященных инструментам установления соответствия между формальными моделями управления доступом и программными реализациями СЗИ. В работе [5] описана общая схема тестирования СЗИ на основе формальной модели управления доступом. В статье [6] более подробно рассматривается вопрос анимации (интерпретации) трасс, которые собираются в ходе исполнения тестов СЗИ. Данная работа концентрируется на источниках требований к инструменту верификации СЗИ и основных сценариях его использования, на проектных решениях, которые были приняты для обеспечения важнейших из требований, и на компромиссах, на которые пришлось пойти для того, чтобы инструмент был достаточно удобен для потенциальных пользователей.

2. Метод верификации СЗИ и требований к нему на основе формальной модели

Первая проблема, которую предстоит решить для получения промышленного инструмента верификации СЗИ с использованием формальной модели – выбор методов верификации как СЗИ, так и самой модели. Как уже было упомянуто, это необходимо делать, во-первых, в соответствии со Стандартом, во-вторых, с учетом потенциального промышленного использования полученного решения.

Часть 1 Стандарта [1] дает общие положения и содержит формулировки основных понятий. Центральным является понятие формальной модели управления доступом. Эта модель описывает, как принимается решение о предоставлении доступа. Средства защиты информации должны реализовать эту модель, то есть предоставлять или не предоставлять

доступ в строгом соответствии с моделью. В качестве одного из возможных языков для разработки формальной модели предлагается язык Event-B [7]. Он формальный, компактный, и при этом для этого языка есть средства верификации корректности моделей уровня, близкого к промышленному.

Части 2 и 3 [2-3] описывают подробнее этапы разработки и верификации формальной модели. Часть 4 [4] посвящена тому, как можно установить соответствие между формальной моделью управления доступом и программной реализацией СЗИ. Речь идет о том, что необходимо обосновать, что программная реализация СЗИ выполняет правила предоставления доступа из формальной модели.

С технической точки зрения Части 2 и 3 отличаются от Части 4 тем, что для первых имеется инструментарий (среда Rodin [8]), документация и примеры для обучения. Всё это необходимо для освоения процессов разработки формальных моделей на практике. Этого нельзя сказать о Части 4, подобных инструментов промышленного уровня нет в практике.

Часть 4 Стандарта дает несколько возможных путей для выполнения верификации СЗИ на соответствие формальной модели управления доступом. Прежде всего, необходимо проанализировать эти рекомендации применительно к существующим задачам, в частности, к верификации СЗИ в операционных системах.

Во-первых, верификация может проводиться на модульном и на системном уровнях. Модульная верификация уместна в случае, когда СЗИ действительно является модулем. Тогда все действия, связанные с контролем доступа, сосредоточены в этом модуле, а структура интерфейса модели соответствует структуре модуля. На системном уровне, когда функции защиты информации распределены по всей системе, модель описывает поведение системы в целом.

Во-вторых, верификацию можно выполнять статическими и динамическими методами, а также аналитически. При выборе метода в реальных промышленных условиях необходимо соблюдать баланс между надежностью и трудозатратами. Так, классические формальные методы надежны, но требуют высокой квалификации персонала и затрат, порой кратно превышающих затраты на разработку. Тестирование системы в динамике требует гораздо меньше ресурсов, но без соблюдения дополнительных требований к процессу является недостаточно надежным.

Опыт моделирования и верификации СЗИ в операционных системах показал, что модульный подход для них практически невозможен. При этом для системного уровня статические и аналитические методы представляются довольно трудоемкими. Баланс между трудоемкостью и качеством верификации может дать упомянутый в Стандарте метод динамической верификации.

Процесс динамической верификации схож с тестированием на основе формальных моделей, что обеспечивает относительную легковесность метода. При этом формальная модель обеспечивает точные формулировки, полноту и непротиворечивость требований. Также надежность верификации будет выше, если тестовый оракул будет построен автоматически на основе модели. Полнота тестирования, согласно Стандарту, должна проверяться в том числе покрытием по модели.

Исходя из приведенных соображений, в качестве промышленного инструмент верификации СЗИ на основе модели имеет смысл использовать инструмент динамической верификации СЗИ на системном уровне. Опыт использования подобной методики отражен в статье [9] и монографии [10]. АНИС является попыткой реализовать именно такой инструмент.

Для реализации инструмента АНИС необходимо решить множество исследовательских задач, которые будут в статье далее рассмотрены.

На рис. 1 представлена схема процесса верификации СЗИ с использованием формальной модели, где в качестве языка моделирования используется Event-B.

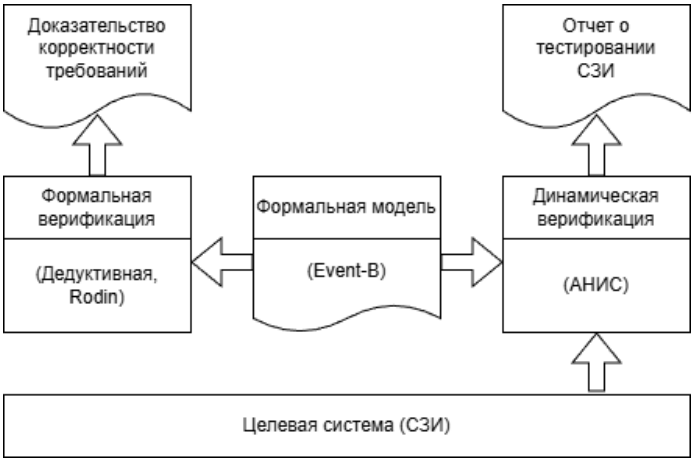

Рис. 1. Схема верификации СЗИ и требований с использованием Event-B.
Fig. 1. Scheme of information security system and requirements verification using Event-B.

3. Система динамической верификации СЗИ

При реализации инструмента динамической верификации необходимо решить множество задач. Одной из них является построение непосредственно процесса динамической верификации, который, во-первых, будет «под капотом» инструмента, а во-вторых, может определять некоторые выдвигаемые к инструменту требования.

Важным элементом системы динамической верификации является монитор. Это программа, которая имеет доступ к тестируемой системе и наблюдает за ее действиями, перехватывая выполняемые операции и их результаты. При динамической верификации система со СЗИ работает в контролируемой среде или без нее. Система выполняет специализированный тест, пакет специального программного обеспечения или действия инициируются пользователем-оператором. Монитор собирает данные о наблюдаемом поведении системы, которые затем сравниваются с описанным в модели ожидаемым поведением.

Есть несколько принципов работы мониторов (рис. 2). Пассивный монитор (вверху рисунка) запоминает действия СЗИ и не вмешивается в ее работу. Монитор собирает трассу исполнения, которая анализируется отдельно (во время работы СЗИ или после нее). Проверкой трассы в этом случае занимается отдельный компонент, называемой программой-оракулом. Активный монитор (внизу рисунка) проверяет, правильно ли СЗИ принимает решение о предоставлении или запрете доступа. Если монитор принимает решение, что СЗИ себя ведет неправильно, он может прервать работу СЗИ. Кроме того, монитор может вести журнал с выполненными проверками СЗИ.

Основной проблемой при разработке мониторов является потеря в эффективности тестируемой системы [11]. Некоторые ошибки в работе тестируемой системы могут проявляться только при специфичном по времени выполнении действий, так что включение неэффективного монитора может помешать обнаружению таких ошибок. Хотя модели управления доступом могут быть достаточно сложными, для активных мониторов приходится ограничиваться простыми проверками в угоду эффективности. Поэтому на практике динамическая верификация, если ее проводить с активным мониторингом, выполняется только для части требований из политики управления доступом.

Из этого сравнения следует основное требование к инструменту динамической верификации промышленного уровня – это необходимость ***интегрировать эффективные мониторы и формальные модели управления доступом*** [12]. Первые обеспечат сбор трасс для проверки,

а вторые – проверку корректности, и оба этих инструмента должны работать с едиными форматами данных.

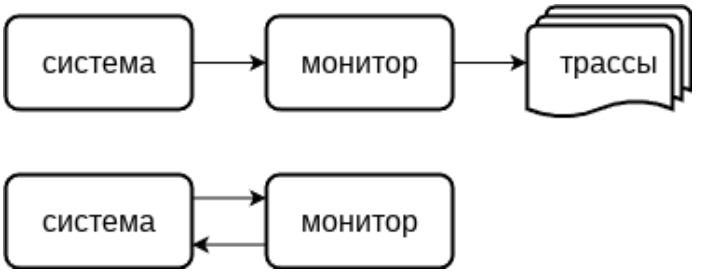

Рис. 2. Виды мониторов (сверху – пассивный, снизу – активный).
Fig. 2. Types of monitors (a top one is passive; a bottom one is active).

Если формальная модель содержит достаточно сложные проверки корректности, имеет смысл в качестве первого шага рассматривать пассивные мониторы. Поэтому мы остановимся на системе динамической верификации именно с пассивным монитором. Вопросы программирования мониторов в данной статье не рассматриваются, главное, что монитор настроен, и он умеет собирать трассы исполнения.

Как уже говорилось, в системе динамической верификации с пассивным монитором необходимы и другие компоненты, в частности, программа-оракул. Подробнее рассмотрим структуру системы динамической верификации.

Монитор фиксирует последовательность операций СЗИ с их аргументами и результатами (в частности, была ли операция успешна и доступ предоставлен или неуспешна и доступ отклонен). Фиксируемая информация должна быть полностью воспроизводима, то есть кроме последовательности операций содержать описание начального состояния тестируемой системы. Последовательность операций и начальное состояние будем называть системной трассой.

Как только системная трасса готова целиком, ее нужно проверить на формальной модели. Среди разных видов формальных моделей здесь рассматриваются те, которые моделируют интерфейсы системных операций [10]. Такие модели обладают состоянием, операциями (событиями), которые меняют состояние. События содержат параметры и охранные условия, то есть условия, при которых событие разрешено выполнять. Должно быть соответствие между событиями и системными операциями. Параметрами события (как минимум) должны быть аргументы системных операций и результаты системных операций, то есть какие изменения операция оказала на систему, на ее сущности. Еще один результат системных операций – успешность операции, то есть, по сути, булевское значение, предоставляет ли доступ система в данной операции. Аналогом этого в формальной модели будет истинность всех охранных условий события.

Чтобы проверить, правильно ли СЗИ приняла решение об успешности операции, нужно проверить охранное условие соответствующего события. Для этого необходимо построить параметры события по аргументам и результатам системной операции из системной трассы. Системные данные и модельные данные могут отличаться по структуре, и автоматическое преобразование будет невозможно. Тогда это преобразование необходимо запрограммировать вручную. Получается компонент-медиатор, адаптирующий системную трассу к формальной модели. Полученную «модельную» трассу проверяет компонент-оракул.

Таким образом, еще одним требованием к разрабатываемому инструменту является ***возможность реализации в нем описанной системы верификации*** (рис. 3). Данная система поддерживает Стандарт, и была экспериментально испытана в ходе работ по верификации

СЗИ в ядре ОС AstraLinux [13]. Отметим, однако, что это далеко не единственный возможный подход: во-первых, система верификации с активным монитором уже будет выглядеть иначе, во-вторых, даже с пассивным монитором можно предложить другие технические решения и реализовать их в рамках того же самого или отдельного набора инструментов.

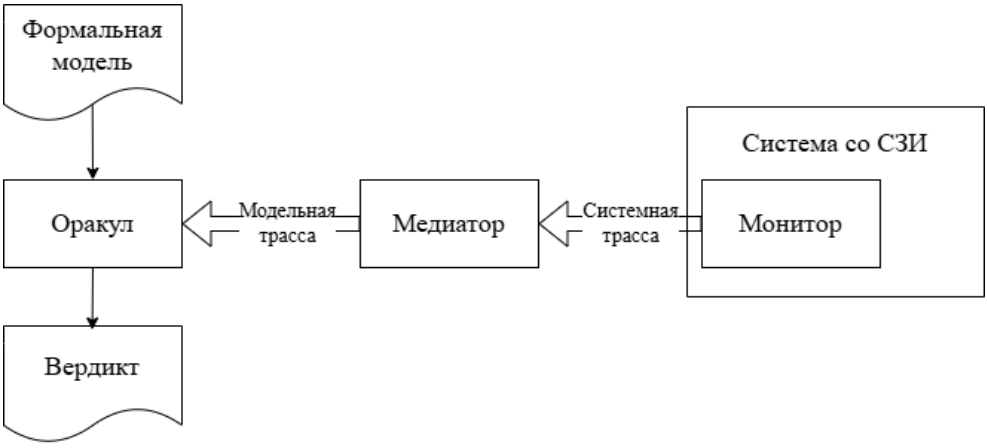

Рис. 3. Структура системы динамической верификации.
Fig. 3. Structure of a runtime verification system.

4. Пользовательские процессы верификации

Еще одной важной задачей при разработке инструмента является анализ возможных сценариев работы пользователей. Это могут быть следующие группы пользователей:

- разработчики формальной модели;
- тестировщики, тестирующие СЗИ на регулярной основе;
- инженеры, готовящие СЗИ к сертификации;
- аттестационные организации, участвующие в сертификации.

Опишем сценарий работы в инструменте пользователя, который работает с формальной моделью в процессе подготовки СЗИ к сертификации:

1. выбрать вид монитора из поддерживаемых инструментом;
2. настроить интеграцию с монитором: указать, откуда брать трассы и в какой формат их надо преобразовать;
3. выбрать язык для формальной модели из поддерживаемых инструментом;
4. написать текст модели или импортировать из другой среды, внести необходимо изменения в текст модели, чтобы инструмент мог пользоваться моделью как оракулом;
5. разработать медиатор: реализовать трансляцию системных данных в модельные, реализовать трансляцию системных операций в модельные события (возможно, поддерживая набор данных – состояние медиатора);
6. выполнить проверку системных трасс, как только инструмент распознает их наличие;
7. проанализировать трассы со статусом ошибки и провести цикл отладки модели, медиатора и настроек интеграции с монитором, ошибки в самой системной трассе сообщить тем, кто отвечает за монитор и СЗИ;

8. проанализировать достигнутое покрытие по элементам формальной модели и составить предложения по дополнительным системным трассам для увеличения покрытия.

Этот сценарий является наиболее полным, и разные его части на практике могут выполняться разными специалистами (например, разработчиком модели и специалистом по тестированию).

Гораздо более простым, но не менее важным сценарием является работа пользователя, выполняющего регулярное тестирование:

1. получить готовый проект с моделью, медиатором, а также все необходимое для сбора трасс;
2. запустить сбор трасс, проверку системных трасс, генерацию отчета;
3. проанализировать отчет и сообщить о найденных расхождениях трассы с моделью и о неполноте в отчете о покрытии.

Регулярное тестирование должно быть в максимальной степени автоматизировано. Роль пользователя при этом сводится к анализу отчетов о найденных ошибках и сведений о полноте покрытия.

Таким образом, от инструмента также требуется ***поддержка описанных пользовательских сценариев***.

5. Инструмент АНИС

На основе выделенных требований был разработан инструмент АНИС. Инструмент обеспечивает интеграцию с системой мониторинга целевой системы, анализ собранных трасс с использованием формальной модели, а также сбор покрытия по модели.

Для поддержки основных пользовательских сценариев инструмент АНИС предоставляет пользователю как визуальные средства, так и скрипты. Первые удобнее использовать для подготовки СЗИ к динамической верификации, а вторые – для регулярной проверки и перепроверки.

5.1 Выбор визуальных средств

Одним из хорошо зарекомендовавших себя подходов к разработке визуальных средств является использование фреймворка, куда пользователь может установить инструменты под конкретные цели. АНИС в качестве такого фреймворка использует среду VSCode [14]. Данная среда является современной и популярной, она активно разрабатывается и имеет качественные готовые инструменты, расширяющие ее возможности для пользователей-разработчиков. Кроме того, очень важным преимуществом VSCode является наличие качественной документации, упрощающей разработку новых инструментов-расширений.

Обычно VSCode используется для разработки программного обеспечения, но ее использование оправдано и для инструмента динамической верификации. Разработка систем и моделей промышленного уровня и использование формальных моделей в динамической верификации требует подхода, схожего с разработкой программного кода: в этой, казалось бы, аналитической деятельности появляется острая необходимость в средствах, напоминающих средства тестирования и отладки.

Перечислим те возможности визуальных средств, которые будут необходимы пользователю-разработчику при динамической верификации (рис. 4):

1. для разработки формальной модели управления доступом;
2. для разработки программы-медиатора на языке программирования;
3. для отладки программы-медиатора;

4. для подключения к монитору и просмотру трасс;
5. для запуска проверки трасс и просмотра статуса для каждой трассы;
6. для отладки проверки трассы.

Рис. 4. Что должен предоставить пользователь и что – АНИС.
Fig. 4. What must be provided by user and what by ANIS.

5.2 Средства для работы с языками программирования

Из перечисленных сценариев следует, что в процессе подготовки возникает необходимость в разработке программных компонентов (например, медиатора). Это означает, что в составе визуальных средств пользователя необходимо иметь инструменты разработчика для языков программирования. Здесь очень важен правильный выбор языка программирования, уже поддержанного качественными инструментами разработки.

Для проверки последовательности событий (то есть системных трасс), недостаточно просто иметь формальную модель, декларативно описывающую допустимое поведение системы. Необходим программный код, либо интерпретирующий модель, либо имитирующий ее поведение.

Таким образом, классические языки программирования встречаются в цикле тестирования на основе формальной модели как минимум дважды: при разработке программных компонентов и при воспроизведении поведения модели. Если трасса, модель и медиатор написаны на едином языке программирования, это существенно упрощает процесс разработки тестов. Кроме того, использование единого языка программирования позволяет бесшовно отлаживать сразу и формальную модель, и медиатор на трассе.

Отсюда следует, что одним из свойств инструмента динамической верификации будет выбранный язык программирования для разработки программных компонентов. Язык должен быть известным для практиков и иметь невысокий порог вхождения, поскольку предполагается подключение к нему не только программистов-разработчиков, но и аналитиков, ведь весь массив артефактов в рамках динамической проверки должен проходить тщательный анализ. Для АНИС выбор был сделан в пользу языка программирования Python. Расширения VSCode для языка Python (включая средства отладки) активно пользуются практиками и достаточно хороши.

5.3 Работа с формальными моделями

Как уже упоминалось, для анализа трасс на основе формальной модели необходимо обеспечить возможность симуляции поведения модели. Данная проблема является нетривиальной и подробно рассматривается в статье [15]. Из всех существующих методов был выбран подход с генерацией на основе модели кода исполняемой программы [6].

АНИС предоставляет следующие инструменты для работы с формальными моделями:

1. текстовый редактор с привычными возможностями: раскраска текста, подсказки при наборе текста, навигация по модели, синтаксическая проверка налету, проверка типизации на лету;
2. транслятор из языка формальной модели в язык программирования; программа содержит подпрограммы, или методы, для каждого события, которые вычисляют охранные условия; на основе этого АНИС будет проверять событие в трассе и строить покрытие по элементам модели.

На данный момент в рамках АНИС реализовано расширение для языка Event-B. Поскольку для разработки моделей на Event-B активно используется среда Rodin, то расширение в АНИС содержит команды для импорта и экспорта модели в формат среды Rodin. Это значит, что можно вести разработку одной и той же модели и в Rodin, и в АНИС, конвертируя между разными форматами. Это дает большую свободу разработчика по анализу моделей как дедуктивными, так и динамическими методами.

Возможности текстового редактора Event-B в АНИС отличаются от возможностей в Rodin. Пользователи Rodin отмечают сложности в наборе текста крупных моделей, а также слабые средства навигации по формальной модели. Эти проблемы решены в текстовом редакторе моделей в инструменте АНИС.

Транслятор из языка моделирования в язык программирования генерирует на основе Event-B модели набор подпрограмм на языке Python, которые могут использоваться при тестировании в качестве элементов оракула. Модельное состояние системы служит основой для соответствующего класса, а модельные события – для функций, вычисляющих результаты проверок охранных условий события и изменяющих модельное состояние. Так, очередное событие из трассы с набором присутствующих в трассе параметров события может оказаться:

- применимым (все охранные условия истинны, тогда операция в тестируемой системе должна завершиться успехом);
- неприменимым (хотя бы одно охранное условие ложно, тогда операция в тестируемой системе должна завершиться неудачей);
- ошибочным (часть охранных условий истинна, а оставшиеся невозможно вычислить из-за недостающих в трассе параметров события, это может означать ошибку в процессе формирования трассы).

5.4 Интеграция мониторов и формальных моделей

Одной из основных задач инструмента является интеграция мониторов и формальных моделей. Так как формальная модель пишется на определенном языке моделирования, задача ее интеграции ограничивается поддержкой соответствующего языка. Формат же системных трасс не определен, что на первый взгляд, усложняет задачу интеграции. Однако при ближайшем рассмотрении оказывается, что написание мониторинга сложной системы с нуля является сложной, а главное, необязательной задачей. Так, в случае ядра Linux будет проще воспользоваться уже существующими механизмами, например, технологией eBPF [16]. Таким образом, АНИС может уметь подстраиваться под ограниченное количество форматов трасс существующих инструментов, а в случае, если пользователь реализует мониторинг полностью своими силами, он может подстроиться под форматы трасс, поддерживаемые в АНИС.

АНИС, как система интеграции мониторов и формальных моделей (рис. 5), предлагает относиться к трассам как к тестам для формальных моделей. Поэтому необходимо определиться с тестовым фреймворком, который должен обеспечивать поддержку списка тестов и их запуска. В АНИС для этого использует pytest [17]. Он также хорошо известен

инженерам-программистам и его поддержка входит в расширение VSCode для языка Python. При помощи расширения можно просматривать список тестов, запускать тесты на выполнение (все или частично) в обычном режиме и в режиме отладки. В контексте АНИС отладка теста является и отладкой формальной модели, ведь она тоже представлена в виде программного компонента на Python и выполняется в рамках тестов.

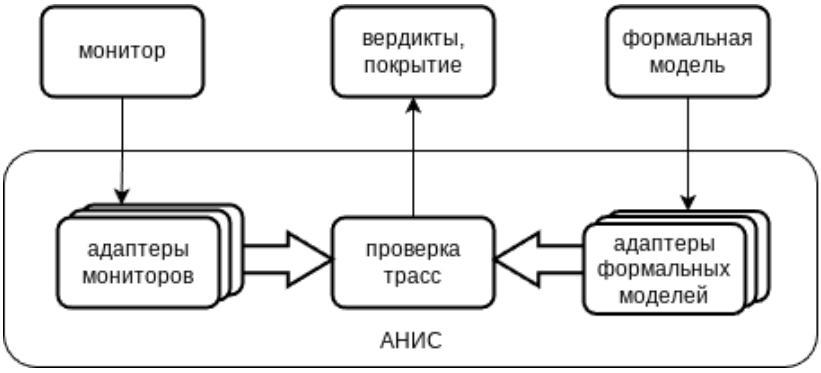

Рис. 5. АНИС интегрирует мониторы и формальные модели.
Fig. 5. ANIS integrates monitors and formal models.

Транслятор системных трасс выполняет функцию адаптеров мониторов. Результатом трансляции системных трасс является набор pytest-тестов. Если при этом запуск тестов на системе с монитором также осуществлялся при помощи pytest, и пользователь знаком с ними, то в результате трансляции системных трасс пользователь увидит уже знакомую структуру тестов. Чтобы адаптер смог восстановить структуру, монитор должен записать в трассу полный идентификатор изначального теста, который будет задействован для повторения структуры сгенерированных тест для трасс.

Транслятор формальной модели в язык программирования выполняет функцию адаптера формальных моделей. Когда проверка трассы запускается в режиме отладки, отладчик приостанавливает выполнение при возникновении ошибки (программной или обнаруженное несоответствие с моделью) в исходном коде трасс, медиатора или в транслированном программном коде для модели. Средства отладки работают одинаково в обоих случаях, так как медиатор, системная трасса, модельная трасса и код модели по факту являются частями одной программы.

5.5 Покрытие по элементам формальной модели

У инструмента динамической верификации могут быть разные группы пользователей и заинтересованных лиц. Это могут быть разработчики формальной модели, разработчики СЗИ, руководители проектов, представители контролирующих организаций. Им важно получать отчеты в понятном им виде, и для разных пользователей эти виды будут различаться.

Сбор покрытия по элементам формальной модели является основным критерием полноты тестирования в рамках динамической верификации. Такой вид покрытия пока остается за рамками общемировой практики, однако задача его реализации является скорее технической. Одновременно с этим покрытием также можно собирать покрытие по исходным текстам СЗИ. Опыт показывает, что для качественной оценки тестового набора требуются оба вида отчетов о покрытии.

Более того, часть 4 Стандарта предписывает собирать такие отчеты о покрытии. В приложении Б к части 4 приводятся рекомендации по оценке покрытия формальных моделей.

В первую очередь это покрытие отдельных охранных условий: каждое охранное условие должно возникнуть в трассах с истинным значением и сложным. Исключения делаются для охранных условий, которые всегда принимают только одно из логических значений. Это охранные условия, фиксирующие типы данных параметров, и охранные условия, истинность которых обусловлена выбором значений параметров в медиаторе. Имеет смысл добавить возможность отмечать охранные условия как имеющие единственное значение.

Рекомендуется еще один вид оценки покрытия формальной модели, имитирующий покрытие типа MC/DC [18]. А именно, для каждого охранного условия, которое может иметь два логических значения, должен быть элемент трассы (событие), в которой это охранное условие – единственное, имеющее ложное значение, среди охранных условий этого же события. Если в результате окажется, что некоторое множество охранных условий не покрыто, это может стать поводом для более тщательного анализа формальной модели: действительно ли эти охранные условия не могут быть ложными по отдельности или это получилось, например, из-за ошибки в модели.

Если охранное условие само является неатомарной формулой, то Стандарт рекомендует проводить аналогичную оценку покрытия отдельных подформул. Для формулы-конъюнкции нужны трассы, где все подформулы истинные, и трассы, где ложная только одна из подформул. Для формулы-дизъюнкции нужны трассы, где все подформулы ложные, и трассы, где истинна только одна из подформул. Отличие этого покрытия от покрытия охранных условий в том, что для покрытия элементов формулы некоторого охранного условия остальные охранные условия этого же события должны быть истинными.

В АНИС были успешно реализованы все перечисленные виды покрытий по модели, в том числе покрытие, имитирующее MC/DC.

6. Ограничения и возможности расширения. Экспериментальная эксплуатация

Приведенная выше схема системы динамической верификации довольно общая. В процессе ее реализации могут возникнуть технические ограничения.

Инструмент АНИС следует этой схеме, тем не менее на данный момент он работает только с формальными моделями, описывающими интерфейс СЗИ (но, например, не с формальными моделями политик безопасности). Это обеспечивает понятные техники разработки компонентов инструмента АНИС. В итоге, инструмент АНИС позволяет обосновывать соответствие формальной модели интерфейса СЗИ и реализации СЗИ. Стандарт требует обосновывать соответствие формальной модели управления доступом и реализации СЗИ. Эта модель более абстрактная, чем формальная модель интерфейсов СЗИ, и часто описывает действия иного плана, чем операции интерфейса СЗИ.

Еще одним ограничением инструмента АНИС является поддержка лишь подмножества языка формального моделирования. А именно, не поддерживаются конструкции, по которым нельзя однозначно построить программный код или этот код не завершается. К первой ситуации относится конструкция «недетерминированное действие». Она позволяет задавать новое значение переменной не в виде выражения, зависящего только от старого значения переменной, а в виде предиката, зависящего от старого и нового значения переменной. Для полноценной поддержки этой конструкции необходимо внедрять техники разрешения предикатов. Пока такая конструкция поддерживается только для специальных случаев предикатов. Подробнее об этом написано в статье [6]. Ко второй ситуации относятся кванторные выражения, в которых отсутствует явное указание на множество значений подкванторной переменной. Тем не менее, опыт написания формальных моделей интерфейсов СЗИ показывает, что эти ограничения не являются существенными, а неразрешенные конструкции могут быть уточнены или заменены на поддерживаемые конструкции.

Несмотря на эти ограничения, с помощью инструмента АНИС успешно удалось воспроизвести процессы динамической верификации с использованием Event-B моделей, применяемые в предыдущих исследованиях. В настоящее время АНИС успешно применяется для динамической верификации СЗИ в реальной ОС (AstraLinux).

В рамках экспериментальной эксплуатации АНИС для верификации AstraLinux, инструмент продемонстрировал успешную работу с моделью, описанной в [13], применялся для расширения и исправления ошибок в модели. Неожиданно АНИС успешно показал себя в роли инструмента для тестирования системы мониторинга: инфраструктура тестовой системы, как единой программы, позволила выявить ошибки, которые без АНИС не получалось воспроизвести.

Однако, разумеется, технология не является исчерпывающей: необходимо также понимать возможности и потребности по расширению АНИС.

Во-первых, выбор базовой платформы (в текущей версии АНИС это VSCode). От платформы требуется наличие возможностей по управлению тестами на языке программирования, а также отладки. Базовая платформа должна позволять добавлять в нее новые редакторы для работы с текстом формальных моделей. Она должна взаимодействовать с редактором, обмениваясь действиями пользователя (по редактированию, по конфигурированию и др.). Этот набор требований является стандартным для IDE, поэтому при необходимости эта задача легко решается.

Во-вторых, выбор языка формального моделирования. Ожидается, что модели на нем имеют схожую структуру с моделями на Event-B, то есть в них есть аналоги событий, у событий есть аналог охранных условий, или предусловий, и есть аналог действий. Эти элементы используются в алгоритме проверки трассы. В противном случае потребуется другой алгоритм проверки и для него нужно будет реализовать плагин-расширение с новым алгоритмом. Кроме того, для нового языка потребуется новый плагин-расширение со средствами редактирования и трансляции в объектно-ориентированный язык программирования. Здесь нужно проанализировать язык моделирования и продумать алгоритм трансляции; определить конструкции, которые невозможно транслировать, и конструкции, дающие неэффективный программный код.

В-третьих, выбор базового языка программирования. Он должен быть объектно-ориентированным, чтобы была возможность переопределения. Для него должен существовать развитый тестовый фреймворк, и поддержка этого фреймворка должна быть в базовой платформе. Пользователи должна быть знакомы с этим языком и в нем должно быть достаточно конструкций для результата трансляции формальной модели.

В-четвертых, интеграция мониторов. Задачи адаптера для монитора известны, поэтому можно разрабатывать сторонние адаптеры и добавлять в базовую платформу как плагин-расширение. АНИС обнаружит новые трассы-тесты и позволит пользователю их проверить.

7. Заключение

Типичными примерами средств защиты информации, которые поддерживают управление доступом к различным информационным объектам, систем являются операционные системы (объектами защиты здесь являются файлы, процессы, порты ввода/вывода и др.) или СУБД (где объектами защиты могут быть таблицы, записи, поля записей и др). В информационных системах ответственного назначения контроль управления доступом должен отвечать требованиям политик информационной безопасности, которые сами по себе должны иметь четкое, однозначно интерпретируемое описание. Строгое и четкое описание требований, особенно, если речь идет о сложной многопользовательской системе, должно быть представлено в формализованном, машиночитаемом виде. Только в этом случае можно гарантировать согласованность, непротиворечивость и однозначность политики

безопасности (корректность описания политики может быть доказана строго математически) и корректность верификации СЗИ при помощи программных инструментов.

Как уже отмечалось требования к процессам и инструментам верификации СЗИ сложных систем ответственного назначения сформулированы в ГОСТ Р 59453.4-2025, для поддержки которого и разрабатывался инструмент АНИС. АНИС можно рассматривать как экспериментальную разработку, которая подтвердила реализуемость требований Стандарта и позволила исследовать потенциальные проблемы, которые могут возникнуть при разработке других инструментов для поддержки верификации и сертификации СЗИ программных систем ответственного назначения, а также определить базовые сценарии использования набор инструментов, поддерживающих реализацию рекомендаций Стандарт.

В связи с этим наряду с базовой функциональностью АНИС оснащен средствами для разработки и отладки как формальных моделей, так тестов – это позволяет пользователю АНИС работать в привычной среде разработки программ и тестов и использовать АНИС в типовых сценариях его эксплуатации. В настоящее время инструмент АНИС проходит опытную эксплуатацию в рамках верификации СЗИ в операционной системе AstraLinux.

Авторы выражают благодарность Д. В. Буздалову, Д. В. Ефремову, В. В. Кулямину и А.В. Хорошилову за замечания и советы по улучшению инструмента АНИС и текста данной статьи.

Список литературы / References

[1]. ГОСТ Р 59453.1. Защита информации. Формальная модель управления доступом. Часть 1. Общие положения: описание стандарта и тендеры. – Введен 2021-06-01 – Москва: Стандартинформ, 2021. / State Std. GOST R 59453.1. Information protection. Formal access control model. Part 1. General principles. Moscow, 2021 (in Russian).

[2]. ГОСТ Р 59453.2. Защита информации. Формальная модель управления доступом. Часть 2. Рекомендации по верификации формальной модели управления доступом. – Введен 2021-06-01 – Москва: Стандартинформ, 2021. / State Std. GOST R 59453.1. Information protection. Recommendations on verification of formal access control model. Part 2. General principles. Moscow, 2021 (in Russian).

[3]. ГОСТ Р 59453.3. Защита информации. Формальная модель управления доступом. Часть 3. Рекомендации по разработке. – Введен 2025-03-31 – Москва: Стандартинформ, 2025. / State Std. GOST R 59453.3. Information protection. Recommendations on verification of formal access control model. Part 3. Recommendations on development. Moscow, 2025 (in Russian).

[4]. ГОСТ Р 59453.4. Защита информации. Формальная модель управления доступом. Часть 4. Рекомендации по верификации средства защиты информации, реализующего политики управления доступом, на основе формализованных описаний модели управления доступом. – Введен 2025-03-31 – Москва: Стандартинформ, 2025. / State Std. GOST R 59453.4. Information protection. Recommendations on verification of formal access control model. Part 4. Recommendations for verification of information security features that implement access control policies based on formal descriptions of the access control model. Moscow, 2025 (in Russian).

[5]. Петренко А.К., Девянин П.Н., Ефремов Д.В., Карнов А.А., Корныхин Е.В., Кулямин В.В., Хорошилов А.В. Динамическая верификация промышленных средств защиты информации на основе формальных моделей управления доступом. Труды ИСП РАН, том 37, вып. 3, 2025 г., стр. 275–288. DOI: 10.15514/ISPRAS–2025–37(3)–19. / Petrenko A.K., Devyanin P.N., Efremov D.V., Karnov A.A., Kornykhin E.V., Kulyamin V.V., Khoroshilov A.V. Runtime verification of industrial information security tools based on formal access control models. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 3, 2025. pp. 275-288 (in Russian). DOI: 10.15514/ISPRAS-2025-37(3)-19.

[6]. Карнов А.А., Корныхин Е.В. Генерация кода исполняемой модели Event-B на языке Python. Труды ИСП РАН, том 37, вып. 6, 2025 г. / Karnov A.A., Kornykhin E.V. Generating Event-B executable model in Python. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 6, 2025.

[7]. Abrial J.-R. Modeling in Event-B: system and software engineering. Cambridge University Press, 2010.

[8]. Abrial J.-R., Butler M., Hallerstede S., Hoang T., Mehta F., Voisin L. Rodin: an open toolset for modelling and reasoning in Event-B. The International Journal on Software Tools for Technology Transfer, vol. 12. Springer, 2010, pp. 447–466. DOI: 10.1007/s10009-010-0145-y.

[9]. Ефремов Д.В., Копач В.В., Корныхин Е.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы. Труды ИСП РАН, 2021, том 33, вып. 6, стр. 15–26. DOI: 10.15514/ISPRAS-2021-33(6)-2. / Efremov D.V., Kopach V.V., Kornykhin E.V., Kulyamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Runtime verification of operating systems based on abstract models. Proceedings of ISP RAS, 2021, vol. 33, no. 6, pp. 15–26 (in Russian). DOI: 10.15514/ISPRAS-2021-33(6)-2.

[10]. Девянин П.Н., Ефремов Д.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Моделирование и верификация политик безопасности управления доступом в операционных системах. Горячая линия – Телеком, Москва, Россия, 2019. / Devyanin P.N., Efremov D.V., Kulyamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Modeling and verification of access control security policies in operating systems. Moscow, Russia: Hotline-Telecom, 2019 (in Russian).

[11]. De Oliveira D.B., Cucinotta T., De Oliveira R.S. Efficient Formal verification for the Linux kernel. In BSoftware Engineering and Formal Methods: 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings 17, pp. 315–332. Springer, 2019. DOI: 10.1007/978-3-030-30446-1_17.

[12]. Efremov D., Shchepetkov I., Runtime verification of Linux kernel security module. In International Symposium on Formal Methods, 2019, pp. 185–199. Available at: https://arxiv.org/pdf/2001.01442

[13]. Девянин П.Н., Жиляков С.С., Смирнов А.И. Тестирование подсистемы безопасности ОС Astra Linux на основе формализованного описания модели управления доступом. Труды ИСП РАН, том 37, вып. 6, часть 2, 2025, стр. 21–36. DOI: 10.15514/ISPRAS–2025–37(6)–17. / Devyanin P.N., Zhiliakov S.S., Smirnov A.I. Testing the Astra Linux OS security subsystem based on a formalized description of the access control model. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 6, part 2, 2025. pp. 21-36 (in Russian). DOI: 10.15514/ISPRAS-2025-37(6)-17

[14]. Del Sole A., Visual Studio Code Distilled. Springer, 2019. DOI: 10.1007/978-1-4842-9484-0.

[15]. Карнов А.А. Проблема неопределенности в анализе трасс на основе высокоуровневых моделей в контексте динамической верификации. Труды ИСП РАН, 2024, том 36, вып. 4, стр. 169–182. DOI: 10.15514/ISPRAS-2024-36(4)-13 / Karnov A.A. Uncertainty Problem in High-Level Model-Based Trace Analysis as Part of Runtime Verification. Proceedings of the ISP RAS, 2024, vol. 36, no. 4, pp. 169–182. (In Russian). DOI: 10.15514/ISPRAS-2024-36(4)-13

[16]. Gregg B. BPF Performance Tools. Addison-Wesley Professional, 2019, 880 p

[17]. Okken B. Python Testing with pytest. Pragmatic Bookshelf, 2022.

[18]. Chilenski J.J. Investigation of three forms of the modified condition decision coverage (mcdc) criterion. Technical Report, United States. Federal Aviation Administration. Office of Aviation Research, 2001.

Информация об авторах / Information about authors

Алексей Александрович КАРНОВ – научный сотрудник ИСП РАН, преподаватель НИУ ВШЭ. Научные интересы: формальные спецификации, верификация и тестирование, статический и динамический анализ.

Aleksei Aleksandrovich KARNOV – researcher at ISP RAS, lecturer at the National Research University Higher School of Economics. Research interests: formal specifications, verification and testing, static and dynamic analysis.

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 – Cand. Sci. (Phys.-Math.), Associate Professor of system programming departments at the Moscow State University and Senior Researcher at ISP RAS. Research interests: formal deductive verification, model-based testing.

Александр Константинович ПЕТРЕНКО – профессор, доктор физико-математических наук, заведующий отделом Технологий программирования ИСП РАН, профессор кафедр Системного программирования ВМК МГУ и ФКН НИУ ВШЭ. Его научные интересы

включают формальные методы программной инженерии, языки спецификаций и моделирования, их применение для поддержки разработки и верификации программного обеспечения.

Alexander Konstantinovich PETRENKO – Dr. Sci. (Phys.-Math.), Prof., Head of Software Engineering Department at the Ivannikov Institute for System Programming, Russian Academy of Sciences, Professor of the Department of System Programming, Faculty of Computational Mathematics and Cybernetics, Moscow State University and the Faculty of Computer Science, National Research University Higher School of Economics. His research interests include formal methods of software engineering, specification and modeling languages, and their use in software development and verification.