DOI: 10.15514/ISPRAS-2026-38(4)-14

Синтез программного обеспечения и отслеживание элементов принятия решений в беспилотных роботизированных системах

С.Е. Кондратьев, ORCID: 0000-0002-7028-9407 <razthepsycho@yandex.ru>
Н.В. Казюра, ORCID: 0009-0002-3274-0444 <nikolai.cazyura@yandex.ru>
В.Н. Мещеряков, ORCID: 0000-0003-2887-3703 <mesherek@yandex.ru>

Липецкий государственный технический университет,
Россия, 398055, г. Липецк, ул. Московская, д. 30.

Аннотация. Элементы принятия решений в роботизированных системах, создаваемые вручную, содержат потенциальные ошибки, которые зачастую остаются незамеченными на этапе разработки и проявляются лишь при реальном развертывании системы. Такие дефекты могут привести к серьезным последствиям, включая повреждение робототехнического оборудования и угрозу безопасности окружающей среды, в том числе людей, находящихся в зоне функционирования робота. Для минимизации издержек на этапе проектирования и одновременного усиления гарантий безопасной эксплуатации беспилотных систем предлагается подход, основанный на моделировании элементов принятия решений на абстрактном концептуальном уровне с последующим автоматическим преобразованием этих моделей в исполняемый программный код. Дополнительно реализуется механизм непрерывного отслеживания корректности работы данных элементов в режиме реального времени. В рамках исследования представлены два специализированных программных инструмента: первый осуществляет компиляцию концептуальных моделей элементов принятия решений в готовый к выполнению код, второй автоматически создает системы мониторинга времени выполнения на основе разработанных моделей. Эффективность предложенных инструментов продемонстрирована в ходе симуляционных экспериментов, подтверждающих преимущества интеграции модельно-ориентированной разработки, автоматической генерации кода и динамического мониторинга.

Ключевые слова: автономные роботы; генерация кода; мониторинг времени выполнения; деревья поведения; конечные автоматы; модельно-ориентированная разработка; операционная система ROS 2.

Для цитирования: Кондратьев С.Е., Казюра Н.В., Мещеряков В.Н. Синтез программного обеспечения и отслеживание элементов принятия решений в беспилотных роботизированных системах. Труды ИСП РАН, том 38, вып. 4, часть 1, 2026 г., стр. 257–270. DOI: 10.15514/ISPRAS–2026–38(4)–14.

Software synthesis and decision-making element tracking in unmanned robotic systems

S.E. Kondratyev, ORCID: 0000-0002-7028-9407 <razthepsycho@yandex.ru>
N.V. Kazyura, ORCID: 0009-0002-3274-0444 <nikolai.cazyura@yandex.ru>
V.N. Meshcheryakov, ORCID: 0000-0003-2887-3703 <mesherek@yandex.ru>

Lipetsk State Technical University,
30, Moskovskaya st., Lipetsk, 398055, Russia.

Abstract. Decision-making elements in robotic systems developed manually contain potential errors that often remain undetected during the development phase and manifest only upon actual system deployment. Such defects can lead to serious consequences, including damage to robotic equipment and threats to environmental safety, including people located in the robot's operational zone. To minimize design phase costs while simultaneously strengthening safety guarantees for unmanned systems operation, an approach is proposed based on modeling decision-making elements at an abstract conceptual level with subsequent automatic transformation of these models into executable software code. Additionally, a mechanism for continuous correctness monitoring of these elements in real-time is implemented. The research presents two specialized software tools: the first compiles conceptual models of decision-making elements into ready-to-execute code, while the second automatically generates runtime monitoring systems based on the developed models. The effectiveness of the proposed tools is demonstrated through simulation experiments, confirming the advantages of integrating model-driven development, automatic code generation, and dynamic monitoring.

Keywords: autonomous robots; code generation; runtime monitoring; behavior trees; finite state machines; model-based development; ROS 2.

For citation: Kondratyev S.E., Kazyura N.V., Meshcheryakov V.N. Software synthesis and decision-making element tracking in unmanned robotic systems. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 4, part 1, 2026, pp. 257-270 (in Russian). DOI: 10.15514/ISPRAS-2026-38(4)-14.

1. Введение

Несмотря на значительный технологический прогресс в робототехнике и искусственном интеллекте, проблема принятия автономных решений в динамичных и неструктурированных средах при сохранении безопасных и защищенных операций по-прежнему актуальна [1]. Более того, с этой задачей сталкиваются команды разработчиков, в которых участвуют специалисты, различающиеся техническим образованием и уровнем опыта в проектировании и реализации управляющего программного обеспечения. Критическим элементом зачастую оказывается уровень делиберации – совокупность компонентов, предназначенных для формирования поведения робота с учётом его миссии и текущего состояния среды. Традиционный подход к проектированию, реализации и интеграции компонентов делиберации на основе кода подвержен ошибкам, которые могут не выявляться при симуляции и тестировании и могут быть обнаружены только после развертывания, где они способны вызвать неэффективное или нежелательное поведение. Именно для решения таких проблем исследователи прилагают всё больше усилий к разработке методологий, способных поддержать проектирование и реализацию архитектур управления, гарантирующих ожидаемое качество обслуживания, а также заявленные требования безопасности и защищённости [2-3]. Данное исследование продолжает эту работу, рассматривая подход модельно-ориентированного проектирования (МОП).

В частности, предполагается, что компоненты делиберации моделируются на концептуальном уровне с использованием хорошо известных абстракций, таких как конечные автоматы (КА) и деревья поведения (ДП), и что реализации компонентов взаимодействуют между собой и с другими уровнями архитектуры управления через чётко определённые протоколы и общее промежуточное программное обеспечение – в данном

случае ROS 2 (Robot Operating System) [4]. В этом контексте основной исследовательский вопрос заключается в том, можно ли облегчить разработку компонентов делиберации и в то же время повысить уверенность в их корректном поведении, не создавая излишней нагрузки на разработчиков. Представлены два инструмента и их апробация на практическом примере, чтобы показать, что возможно совместить сокращение времени разработки и повышение уверенности в реализациях: Модель-в-Код (M2K), который автоматически компилирует концептуальные модели компонентов делиберации в исполняемый код, совместимый с ROS 2; и Мониторинг-ОНлайн (МОНИТОР), который генерирует мониторы времени выполнения из моделей для повышения уверенности в соответствии сгенерированного кода их спецификации. МОНИТОР может осуществлять мониторинг свойств и моделей. Первые представляют собой набор условий, выраженных на формальном языке, которым должны удовлетворять данные, передаваемые через каналы связи в архитектуре (см. примеры свойств в разделе 6, табл. 1). Для вторых МОНИТОР проверяет, что отслеживаемые компоненты в архитектуре ведут себя в соответствии с заданной моделью. Тестируется комбинация M2K и МОНИТОР на симулированном, но реалистичном сценарии использования, где демонстрируется, что, во-первых, M2K генерирует код, который не вызывает нежелательного или неэффективного поведения во время выполнения, во-вторых, МОНИТОР способен обнаруживать различные неисправности, внедрённые в реализацию компонентов делиберации, и, в-третьих, мониторы моделей работают без чрезмерных накладных расходов даже в сочетании с мониторами свойств. Кроме того, показывается, что в некоторых случаях мониторы моделей – которые получаются «бесплатно» из спецификации компонентов – способны обнаруживать проблемы, которые мониторы свойств могут не перехватить.

2. Предварительные сведения

2.1 Контекстно-осведомленная верифицируемая и адаптивная динамическая делиберация

Инструменты и методы, представленные в данной работе, представляют собой контекстно-осведомленную верифицируемую и адаптивную динамическую делиберацию, целью которой является разработка программного инструментария, помогающего разработчикам проектировать и создавать полностью верифицированные системы делиберации роботов [3]. Цель – расширить возможности роботов для надёжного и безопасного выполнения сложных задач в неструктурированных средах посредством автономной и неконтролируемой адаптации к среде и операционному контексту путём разработки когнитивных возможностей делиберации, обеспечивающих безопасную работу робота в течение длительных периодов времени без вмешательства человека. В работе эталонная архитектура состоит из трёх уровней:

- **Уровень делиберации**, который оркестрирует различные навыки для получения желаемого поведения, например, проводит группу посетителей по музею и представляет им некоторые произведения искусства.
- **Уровень навыков**, который реализует базовые возможности робота, например, определяет, заинтересованы ли посетители или нет.
- **Функциональный уровень**, который непосредственно взаимодействует с аппаратным обеспечением, например, навигация, распознавание речи, зрение и обнаружение объектов.

Эти уровни должны быть связаны через промежуточное программное обеспечение для обмена данными и координации друг с другом.

2.2 Деревья поведения

Недавние исследования [5-6] подтверждают, что деревья поведения (ДП) [7] остаются широко используемым инструментом в сообществе робототехники для модельно-ориентированного проектирования политик делиберации. Рассматриваются следующие типы узлов и связанный с ними графический синтаксис для ДП:

- **Корневой узел**, обозначаемый как «∅».
- **Управляющие узлы** (выделены красным цветом): Реактивная последовательность, обозначаемая как «R→»; Реактивный резерв, обозначаемый как «R?»; Резерв, обозначаемый как «?».
- **Инвертор**, обозначаемый как «≠» (выделен синим цветом).
- **Узел условия**, представленный овалом (выделен жёлтым цветом).
- **Узел действия**, представленный прямоугольником со скруглёнными углами (выделен светло-зелёным цветом).
- **Поддерево**, представленное прямоугольником (выделено зелёным цветом).

Ввиду того что ДП были введены в практику робототехники сравнительно недавно, их семантика ещё не стандартизирована для робототехнических задач. Хотя попытки сделать семантику выполнения ДП более формальной и явной предпринимались в работах [8-9], легко обнаружить несоответствия в реализации различных библиотек [10]. В данной работе рассматривается семантика, реализованная библиотекой BehaviorTree.cpp, на которой основаны реализации. Чтобы понять, как работают ДП, на рис. 1 (слева) представлено ДП с корневым узлом и единственным дочерним узлом реактивной последовательности; этот узел имеет три дочерних элемента, которые соответствуют поддеревьям. На рис. 1(справа) отображено ДП, состоящее из узла реактивного резерва с двумя дочерними элементами: узлом условия («BatteryLevel») и узлом действия («Alarm»).

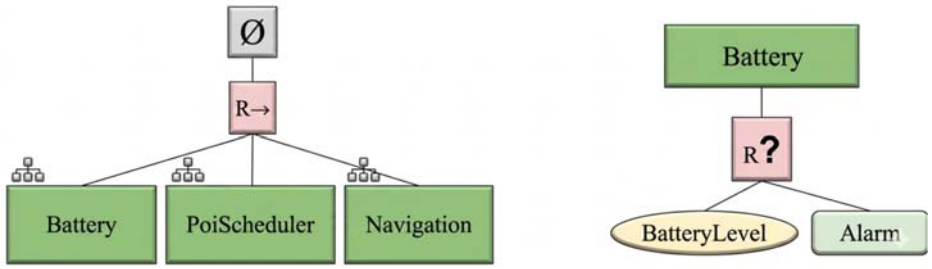

Рис. 1. Примеры ДП в компактном представлении.
Fig. 1. Example of BTs in compact representation.

Выполнение начинается с подачи сигнала тика в корневой узел, после чего этот сигнал последовательно распространяется вниз по структуре дерева. Предполагается, что тики отправляются периодически (например, раз в секунду) и что ДП возвращает результат в течение двух последовательных тиков. Внутри ДП узел условия может ответить либо «успех», если указанное условие истинно, либо «неудача», если указанное условие ложно. Узлы действий также могут ответить «выполняется», если соответствующее действие всё ещё выполняется. Семантика выполнения управляющих узлов зависит от их природы. Узел резерва отправит тик первому дочернему элементу и, при успехе дочернего элемента, вернёт «успех», тогда как при неудаче дочернего элемента будут отправлены тики последующим узлам; узел последовательности отправит тик первому дочернему элементу и, при успехе дочернего элемента, отправит тики последующим узлам. Таким образом, узел резерва

возвращает «неудачу» только если все дочерние элементы завершились неудачей, тогда как узел последовательности возвращает «успех» только если все дочерние элементы завершились успешно. В обоих узлах, если один из дочерних элементов возвращает «выполняется», то на следующей итерации именно этот дочерний элемент получает тик первым, в то время как остальные сохраняют результат, возвращённый на предыдущем тике. Реактивная последовательность действует как последовательность, но если предыдущий результат дочернего узла был «выполняется», она снова отправит тик всем своим дочерним элементам.

2.3 Конечные автоматы

Конечные автоматы (КА) широко используются для моделирования политик и представления как абстрактных, так и конкретных моделей функциональных и управляющих этапов программной архитектуры. КА, включающие вероятностные или временные переходы, также могут представлять абстрактные модели аппаратных компонентов или среды. Автоматы представлены, используя подмножество стандарта SCXML [11], исключая только иерархические автоматы и сценарии кода. Эти возможности могут быть проблематичными: во-первых, семантика иерархических КА несколько непоследовательна в различных работах в литературе [12]. Следовательно, реализация зависит от библиотеки, что делает мониторинг модели непоследовательным. Во-вторых, сценарии кода (на основе ECMAScript) требуют назначения семантики фрагментам кода, что может стать проблематичным для больших фрагментов кода.

Используется следующее представление SCXML:

- **Состояние**: состояние машины SCXML;
- **Событие**: событие, получаемое КА, может быть внутренним, то есть исходящим от самого КА, или внешним, то есть исходящим от источника, внешнего по отношению к КА;
- **Переход**: переход из одного состояния в другое, который может быть безусловным или основанным на событии.

События управляются в соответствии со спецификацией SCXML, то есть все они ставятся в очередь и обрабатываются в соответствии с их возникновением. Кроме того, язык был обогащен элементами, специфичными для промежуточного ПО, для получения информации, специфичной для реализации. В частности, для ROS 2 были реализованы «сервисы» и «топики» для указания того, генерирует ли событие сообщение через них или генерируется сообщением. Эти элементы являются фундаментальными для генерации кода и мониторинга, описанных в разделах 3 и 4.

2.4 Промежуточное программное обеспечение

Использование промежуточного ПО для связи между компонентами позволяет распределять их на разных машинах с минимальными усилиями, позволяя архитектуре управления масштабироваться по мере необходимости. В принципе, предлагаемый подход может быть реализован поверх любого промежуточного ПО, но в данной работе предполагается, что модели ДП и КА основаны на ROS 2 [4], коммуникационном промежуточном ПО, широко используемом в сообществе робототехники. ROS 2 в основном состоит из трёх основных интерфейсов:

- **Топики**: основаны на паттерне издатель-подписчик, их коммуникация типа «многие ко многим», что означает, что можно иметь множество издателей и множество подписчиков для каждого топика. Они асинхронны и типизированы.

- **Сервисы**: основаны на паттерне клиент-сервер. Они синхронны и типизированы. Их коммуникация типа «многие к одному», что означает, что множество клиентов могут подключаться к одному серверу.

Действия: представляют собой комбинацию сервисов и топиков и предназначены для длительных вычислений. Действие состоит из двух сервисов: одного для начала вычисления и одного для получения результата. Кроме того, имеется топик, который после начала вычисления предоставляет клиенту обратную связь о статусе вычисления. Это позволяет клиенту продолжать свою работу и получить результат по завершении действия.

3 M2K (Модель-в-Код)

Уровень навыков (раздел 2.1) реализуется с использованием КА на основе языка SCXML (раздел 2.3). В этих моделях события запускают переходы между состояниями, позволяя машине изменять своё поведение и взаимодействовать с внешним миром. Хотя автоматы SCXML могут быть скомпилированы и выполнены, они не могут напрямую взаимодействовать с функциональным уровнем, поскольку эти элементы не определены в SCXML. Это создаёт разрыв между уровнем навыков и функциональным уровнем, требуя интерфейса, который транслирует события SCXML в действующие команды, взаимодействующие с системой в соответствии с выбранным фреймворком. M2K – это инструмент, предназначенный для автоматизации генерации этой интерфейсной системы для каждого автомата.

Инструмент позволяет пользователям сосредоточиться на высокоуровневом проектировании автоматов, автоматизируя генерацию базового интерфейса, ручная разработка которого была бы трудоёмкой и подверженной ошибкам. M2K использует подход на основе шаблонов, принимая на вход модели SCXML и генерируя на выходе соответствующий код C++ для ROS 2, который транслирует события SCXML в интерфейсы ROS 2, как указано в разделе 2.4. Входные файлы включают модель SCXML и XML-файл, описывающий архитектуру системы с указанием определения интерфейсов. Начиная с набора файлов-шаблонов C++, инструмент производит следующий вывод: заголовочный файл C++ и исходный код исполняемого навыка, главный файл, содержащий код обработки событий и соответствующую трансляцию ROS 2, файл конфигурации CMakeLists.txt и XML-файл пакета ROS 2. Выходные данные могут быть собраны непосредственно с помощью CMake без дополнительного редактирования.

4 МОНИТОР (Мониторинг-ОНлайн)

МОНИТОР – это монитор времени выполнения, реализованный на основе ROSmonitoring [13], к которому были добавлены верификация модели и автоматическая генерация монитора.

МОНИТОР принимает ту же спецификацию модели, что и M2K, и обеспечивает генерацию мониторов для свойств и моделей. На данный момент генерация мониторов доступна только для сервисов и топиков с целью расширения также на действия в будущем. Для проверки сообщений, проходящих через каналы, и верификации свойств МОНИТОР опирается на асинхронные средства связи, предоставляемые механизмами публикации-подписки, доступными в ROS 2. Для мониторинга моделей, с другой стороны, он вводит специфические компоненты типа «компонент-посредник», которые перехватывают сообщения, передаваемые сервисами ROS 2. Это вносит определённую степень накладных расходов, которые могут быть снижены с использованием средств интроспекции, доступных в последних дистрибутивах. Реализация мониторинга модели с использованием этих средств находится в процессе разработки. Предварительно ожидается, что переход от синхронного компонента «компонент-посредник» к асинхронному механизму на основе интроспекции позволит существенно сократить накладные расходы мониторов моделей, приблизив их к

показателям мониторов свойств, которые уже используют асинхронную связь и демонстрируют минимальное влияние на время выполнения (см. раздел 6.2). После проверки сообщения пересылаются оракулу. Оракул поддерживает мониторинг через спецификацию свойств и через спецификацию модели. В первом случае оракул проверяет, что отслеживаемые каналы связи удовлетворяют свойству, как описано в [14], и в этом случае можно отслеживать как вызов сервиса, так и ответ сервиса, в зависимости от того, что указано в свойстве. Для мониторинга модели оракул рассматривает КА навыка, созданный на этапе проектирования, и гарантирует, что выполнение реализации навыка – сгенерированной M2K из того же самого КА – соответствует модели. Оракул в настоящее время реализован с использованием библиотеки Apache Commons SCXML для поддержки моделей SCXML. Предлагаемая спецификация модели компилируется в полностью совместимый автомат SCXML, называемый автоматом оракула, который используется во время выполнения для оценки соответствия навыка его спецификации КА. Оракул обрабатывает входы и выходы отслеживаемого навыка и имитирует его поведение, пересылая отслеживаемую коммуникацию автомату оракула, и сравнивает произведённые выходы с отслеживаемыми, оповещая в случае обнаружения какого-либо расхождения. В целом мониторы необходимо настраивать вручную, предоставляя информацию о соответствующих частях системы, которые нужно отслеживать; однако, поскольку вся необходимая информация включена в предлагаемую спецификацию модели, МОНИТОР автоматически генерирует файлы конфигурации монитора.

5 Экспериментальная оценка

5.1 Описание варианта использования

Рассмотрим сценарий, в котором гуманоидный робот проводит экскурсию для посетителей по лаборатории. Для упрощения симуляции речевое взаимодействие не рассматривается. Робот перемещается по лаборатории, чтобы достичь двух заранее определённых точек интереса (ТИ), при этом активно следя за тем, чтобы посетители следовали за ним, останавливаясь, когда они отстают. Он также отслеживает уровень заряда батареи, активируя сигнал тревоги, когда он падает ниже определённого порога. Следует отметить, что представленная экспериментальная оценка проведена исключительно в симуляционной среде; валидация на реальном робототехническом оборудовании планируется в рамках дальнейших исследований.

Согласно архитектуре, определённой в разделе 2.1, есть три уровня. Функциональный уровень не реализуется, и информация имеется только о его интерфейсах; уровень навыков реализуется с помощью КА, как определено в разделе 2.3; а уровень делиберации реализуется с помощью ДП, как определено в разделе 2.2. Вся коммуникация осуществляется через промежуточное ПО ROS 2. Политика робота инкапсулирована в ДП, компактное представление которого в виде поддеревьев показано на рис. 1(а), а расширенная версия – на рис. 2.

Поддерево Battery состоит из управляющего узла реактивного резерва, который отправляет тик листовому узлу «BatteryLevel», который возвращает «неудачу», когда заряд батареи робота ниже порогового значения; в этом случае отправляется тик листовому узлу Alarm, и срабатывает сигнал тревоги. Поддерево «PoiScheduler» обрабатывает установку текущей ТИ через узел резерва, первая и вторая ветви которого относятся к первой и второй ТИ соответственно. Листовые узлы «IsPoiDone» проверяют, достигнута ли определённая ТИ, в то время как узлы «SetPoi» устанавливают определённую ТИ как текущую. Листовой узел «Reset» получает тик по окончании экскурсии для начала новой экскурсии и сброса каждой ТИ. Поддерево «Navigation» проверяет присутствие посетителей и ожидает их, если они отсутствуют. В противном случае отправляется тик листовому узлу действия «GoToPOI», он

возвращает «выполняется», и робот движется к текущему пункту назначения. При достижении ТИ отправляется тик «SetCurrentPOIDone», и эта ТИ помечается.

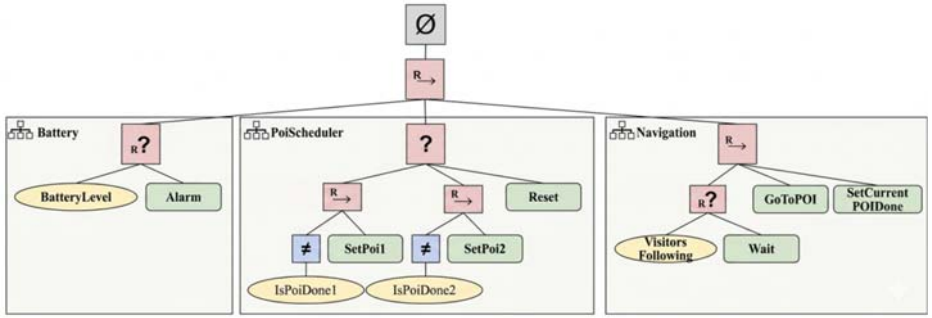

Рис. 2. Расширенная версия представления ДП.
Fig. 2. Expanded version of BTs representation.

5.2 Мониторинг

Рассматриваются навыки «BatteryLevel», «Alarm», «IsPoiDone1», «SetPoi1» для мониторинга модели, поскольку они имеют различную внутреннюю логику и опираются на различные интерфейсы ROS 2, то есть навык «BatteryLevel» опирается только на топики из среды, в то время как другие навыки имеют один или несколько сервисов для связи. Кроме того, доступны некоторые свойства, связанные с общесистемными требованиями. Каждое свойство, как показано в табл. 1, имеет назначенный паттерн, который транслируется в формулу темпоральной логики, из которой может быть сгенерирован монитор свойств – подробности см. в [14]. Запускаются как мониторы моделей, так и мониторы свойств, чтобы проверить конфигурации, в которых мониторы моделей могут дополнять мониторы свойств, обеспечивая при этом ещё большую степень автоматизации. В частности, хотя генерация мониторов свойств требует от пользователей определения конкретных требований, мониторы моделей генерируются из КА, разработанных разработчиками, без дополнительных усилий с их стороны. Для оценки способности системы обнаруживать аномальные обстоятельства в выбранные навыки внедряются ошибки, либо в автомат SCXML, либо в сгенерированный код. Рассматриваются оба метода, потому что они дают нам больше гарантий при оценке надёжности общего подхода. Например, изменяя событие SCXML, мы можем имитировать неправильную трансляцию этого события инструментом M2K. Введены три различные категории неисправностей:

1. **Отсутствие отклика**: навык не отвечает на тик.
2. **Некорректный ответ**: навык отвечает на тик, но не всегда корректно.
3. **Некорректные имена параметров**: навык имеет опечатку, например, в имени события, сервиса или топика.

6 Экспериментальные результаты

6.1 Обнаружение ошибок

Для тестов используется модель единственной неисправности, то есть в каждой конфигурации внедряется одна ошибка, что гарантирует, что это единственное отличие от номинального поведения системы. Выбор данной модели обусловлен необходимостью изолированной оценки способности мониторов обнаруживать каждую категорию неисправностей в отдельности, что обеспечивает однозначную интерпретацию результатов.

Вместе с тем следует отметить, что в реальных условиях эксплуатации возможно одновременное возникновение нескольких неисправностей различных категорий, что может привести к их взаимному маскированию или, напротив, к каскадным эффектам, затрудняющим диагностику. Исследование поведения системы мониторинга при множественных одновременных неисправностях является важным направлением будущей работы. Отслеживаемые свойства показаны в табл. 1, а сводка тестов – в табл. 2.

Табл. 1. Отслеживаемые свойства.
Table 1. Monitored properties.

ИД	Паттерн	Описание свойства
P1	ОТСУТСТВИЕ ДО	Всегда верно, что «сигнал тревоги» не возникает до того, как возникнет «уровень_батареи < 30%»
P2	ОТВЕТ ГЛОБАЛЬНО	Всегда верно, что «сигнал тревоги» отвечает на «уровень_батареи < 30%» в течение времени t
P3	ОТВЕТ ГЛОБАЛЬНО	Всегда верно, что «ТИ_1_выбрана» отвечает на «не ТИ_1_завершена» в течение времени t

Табл. 2. Экспериментальные результаты.
Table 2. Experimental results.

ИД теста	Тип неисправности	Навык	Монитор модели	Монитор свойств
T1	Отсутствие отклика	BatteryLevel	Обнаруживает после тайм-аута	P1 – Нет обнаружений P2 – Обнаружение при уровне батареи < 30% P3 – Не применимо
T2	Отсутствие отклика	SetPoi1	Обнаруживает после тайм-аута	P1, P2 – Не применимо P3 – Нет обнаружений
T3	Некорректный ответ	BatteryLevel	Обнаруживает при первом возникновении	P1 – Обнаруживает при первом возникновении P2 – Нет обнаружений P3 – Не применимо
T4	Некорректный ответ	Alarm	Обнаруживает при первом возникновении	P1, P2 – Нет обнаружений P3 – Не применимо
T5	Некорректный параметр	BatteryLevel	Обнаруживает при первом возникновении	P1 – Нет обнаружений P2 – Обнаружение при уровне батареи < 30% P3 – Не применимо
T6	Некорректный параметр	IsPoiDone1	Обнаруживает при первом возникновении	P1, P2 – Не применимо P3 – Нет обнаружений

Отсутствие отклика (T1-2): в этих случаях навык не отвечает на тик, поэтому ДП зависает в ожидании ответа. Для T1 был модифицирован SCXML навыка «BatteryLevel» для достижения такого поведения. После выполнения монитор модели сразу обнаруживает разницу между выходами навыка и его спецификацией, поскольку последняя имеет дополнительное событие, то есть ответ на тик. МОНИТОР продолжает сравнивать выходы и утверждает, что есть аномалия после тайм-аута. Монитор свойств в этом случае использует в основном P1 и P2. P1 всегда выполняется, поскольку навык «Alarm» никогда не получает тик, поэтому МОНИТОР не обнаруживает аномалию; в случае P2 аномалия обнаруживается, но только после того, как уровень заряда батареи падает ниже 30% от начального значения 100%. В тесте T2 был модифицирован навык «SetPoi1» так же, как «BatteryLevel» в T1. Также

в этом случае монитор модели обнаруживает аномалию так же, как в T1, в то время как монитор свойств её не обнаруживает, потому что монитор устанавливает «POI 1 selected» в true после получения ответа на тик, который никогда не поступает.

Некорректный ответ (T3-4): в этих случаях навык отвечает на тик, но ответ случаен из-за ошибок в автоматах. В частности, в T3 была внедрена неисправность в SCXML навыка «BatteryLevel» для случайного возврата «успеха» или «неудачи». Аналогично, в T4 была внедрена неисправность в SCXML навыка «Alarm», который случайно возвращает «успех» или «неудачу», но также никогда не пересылает сигнал тревоги на функциональный уровень. В обоих тестах монитор модели обнаруживает аномалию при первом возникновении. Монитор свойств P1 обнаруживает неисправность в T3 одновременно с монитором модели, поскольку навык «BatteryLevel» возвращает «неудачу» и ДП отправляет тик сигналу тревоги, когда заряд батареи ещё высок. Монитор P2, напротив, не становится ложным, потому что уровень батареи никогда не падает ниже 30%. В T4 мониторы P1 и P2 никогда не становятся ложными, потому что они проверяют только тик навыка, а не его ответ.

Некорректное имя параметра (T5-6): в этих случаях навык имеет опечатку в имени параметра. В частности, в T5 навык «BatteryLevel» подписан на топик ROS 2 с опечаткой, поэтому он не считывает уровень заряда батареи, сохраняя внутреннюю переменную уровня на значении 100%. В T6 навык «IsPoiDone1» имеет 2 интерфейса, взаимодействующих с функциональным уровнем (из-за компонента-посредника системы МОНИТОР): один отслеживаемый, а другой не отслеживаемый монитором модели. В этом случае навык вызывает интерфейс, который не отслеживается, поэтому монитор модели не получает никакой информации о вызове функционального уровня. В T5 монитор модели обнаруживает аномалию, когда уровень заряда батареи падает ниже 30%. Поскольку монитор модели не может видеть внутреннее состояние переменной и ответ на тик всегда успешный, монитор модели не обнаруживает никакой аномалии. Однако когда уровень батареи падает ниже 30%, навык всегда возвращает успешный ответ, в то время как модель в мониторе возвращает неудачу. Монитор свойств обнаруживает аномалию одновременно с монитором модели со свойством P2, в то время как P1 ничего не обнаруживает, поскольку сигнал тревоги никогда не получает тик. Наконец, в T6 монитор модели обнаруживает аномалию немедленно, в то время как монитор свойств не обнаруживает никакой аномалии, потому что с его точки зрения программа, казалось, работала корректно.

6.2 Добавленные накладные расходы

Чтобы проверить, сколько накладных расходов добавляет предлагаемый монитор, было измерено время тика каждого навыка для каждой конфигурации: K1 – без мониторов; K2 – только мониторы свойств активны; K3 – только мониторы моделей активны; K4 – все мониторы активны.

Как видно из табл. 3, время тика каждого навыка с обоими мониторами увеличивается, как и ожидалось.

Основное различие между мониторами свойств и мониторами моделей заключается в том, что мониторы свойств используют асинхронную связь от ROS 2 вместо синхронного компонента «компонент-посредник». Накладные расходы, добавляемые мониторами моделей, составляют в среднем 200-250 миллисекунд на навык. Это связано с тем, что выполнение тика блокируется в ожидании выполнения мониторов, которые выполняются синхронно с вызовом сервиса. Эти результаты демонстрируют, что совместный мониторинг свойств и моделей осуществим. Добавленные накладные расходы не пренебрежимо малы, но приемлемы в данном контексте, поскольку сосредоточены на мониторинге компонентов на уровне делиберации, время отклика которой больше, чем добавленные накладные расходы. Однако необходимо учитывать, что при применении данного подхода к системам с более жёсткими временными ограничениями, например, в отношении к быстрореагирующим

навигационным модулям, системам предотвращения столкновений или задачам, требующим частоты тиков выше 1 Гц, увеличение времени тика корневого узла с ~30 мс до ~864 мс (в 29 раз) может стать неприемлемым. В таких случаях целесообразно применять мониторинг моделей выборочно, ограничивая его наиболее критичными компонентами, либо использовать асинхронные механизмы мониторинга. Кроме того, ожидается, что большая часть накладных расходов из-за мониторинга моделей может быть снижена путём замены подхода «компонент-посредник» асинхронной проверкой каналов на основе механизмов интроспекции ROS 2. Как следует из данных табл. 3, мониторы свойств (конф. 2), использующие асинхронную связь, увеличивают время тика корневого узла лишь на ~24% (с ~30 мс до ~37 мс), тогда как мониторы моделей (конф. 3) – более чем в 27 раз. Это позволяет предположить, что переход мониторов моделей на асинхронный механизм интроспекции может снизить их накладные расходы на один-два порядка.

Табл. 3. Среднее время тика для каждого навыка и корня ДП.
Table 3. Average tick time for each skill and the root of the BT.

Название навыка	Среднее время тика (мкс)			
	К1	К2	К3	К4
BatteryLevel	3 877	4 103	56 943	65 670
IsPoiDone1	6 906	7 030	314 621	327 816
SetPoi1	6 094	7 663	308 686	333 764
Alarm	4 699	8 151	284 138	298 117
ROOT (корень)	29 904	37 036	810 121	864 467

Примечание: *К1 – без мониторов; К2 – только мониторы свойств активны; К3 – только мониторы моделей активны; К4 – все мониторы активны.*

Note: *K1 – no monitors; K2 – only property monitors active; K3 – only model monitors active; K4 – all monitors active.*

7 Заключение

Экспериментальные результаты предоставляют убедительные доказательства того, что автоматизация генерации реализаций навыков и мониторинг результирующего кода могут эффективно повысить уверенность в корректном поведении компонентов делиберации без увеличения нагрузки на разработчиков. Тесты также демонстрируют, что мониторы моделей могут успешно дополнять мониторы свойств и не создают дополнительной нагрузки на разработчиков, поскольку не требуют от них формулирования конкретных свойств.

Вместе с тем необходимо указать на ряд ограничений данного исследования. Во-первых, экспериментальная оценка проведена исключительно в симуляционной среде. При переносе на реальное робототехническое оборудование следует ожидать дополнительных сложностей, связанных с недетерминированностью задержек сетевой коммуникации ROS 2, шумами и неточностями сенсорных данных, ограниченными вычислительными ресурсами бортовых систем, а также непредсказуемостью динамической среды, что может повлиять как на производительность мониторов, так и на корректность обнаружения неисправностей. Во-вторых, в экспериментах использовалась модель единственной неисправности, что не позволяет оценить поведение системы при множественных одновременных ошибках, которые характерны для реальных условий эксплуатации. В-третьих, накладные расходы мониторов моделей, обусловленные синхронным компонентом-посредником, существенны и требуют дальнейшей оптимизации на основе механизмов интроспекции ROS 2 для применения в системах с жёсткими временными ограничениями. Указанные ограничения определяют приоритетные направления будущей работы: валидация на реальном оборудовании, исследование множественных неисправностей и завершение реализации асинхронного мониторинга на основе интроспекции.

Насколько известно авторам, наиболее близким к предлагаемому подходу является фреймворк, представленный в работах [2, 15], где также рассматривается формальная спецификация и развёртывание программных компонентов робототехнических систем с возможностью автоматического синтеза их формальной модели. Однако ключевое отличие состоит в том, что в этом случае исполняется сама формальная модель, а не традиционная программная реализация, и именно за счёт этого обеспечиваются возможности верификации и мониторинга. Хотя в робототехнике уже предложен и успешно применён широкий спектр решений, основанных на модельно-ориентированном проектировании, офлайн-верификации и мониторинге, большинство таких подходов не объединяют автоматическую генерацию кода из концептуальных моделей, как в M2K, с мониторингом как свойств, так и моделей, как в МОНИТОР, причём обе эти возможности реализуются поверх широко распространённого и поддерживаемого промежуточного ПО ROS 2.

Список литературы / References

[1]. Brunke L., Greeff M., Hall A. W., Yuan Z., Zhou S., Panerati J., Schoellig A. P. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. Annual Review of Control, Robotics, and Autonomous Systems, vol. 5, 2022, pp. 411-444. DOI: 10.1146/annurev-control-042920-020211.

[2]. Li W., Ribeiro P., Miyazawa A. et al. Formal Design, Verification and Implementation of Robotic Controller Software via RoboChart and RoboTool. Autonomous Robots, vol. 48, 2024, art. 14. DOI: 10.1007/s10514-024-10163-7.

[3]. Dust L., Gu T., Mubeen S., Ekström M., Seceleanu C. A Model-Based Approach to Automation of Formal Verification of ROS 2-Based Systems. Frontiers in Robotics and AI, vol. 12, 2025, art. 1592523. DOI: 10.3389/frobt.2025.1592523.

[4]. Portugal D., Rocha R. P., Castilho J. P. Inquiring the Robot Operating System Community on the State of Adoption of the ROS 2 Robotics Middleware. International Journal of Intelligent Robotics and Applications, vol. 9, 2025, pp. 454-479. DOI: 10.1007/s41315-024-00393-4.

[5]. Gugliermo S., Cáceres Domínguez D., Iannotta M., Stoyanov T., Schaffernicht E. Evaluating Behavior Trees. Robotics and Autonomous Systems, vol. 178, 2024, art. 104714. DOI: 10.1016/j.robot.2024.104714.

[6]. Merino-Fidalgo S., Sánchez-Girón C., Zalama E., Gómez-García-Bermejo J., Duque-Domingo J. Behavior Tree Generation and Adaptation for a Social Robot Control with LLMs. Robotics and Autonomous Systems, vol. 194, 2025, art. 105165. DOI: 10.1016/j.robot.2025.105165.

[7]. Ögren P., Sprague C. I. Behavior Trees in Robot Control Systems. Annual Review of Control, Robotics, and Autonomous Systems, vol. 5, no. 1, 2022, pp. 81-107. DOI: 10.1146/annurev-control-042920-095314.

[8]. Serbinowska S. S., Johnson T. T. BehaVerify: Verifying Temporal Logic Specifications for Behavior Trees. In: Software Engineering and Formal Methods (SEFM 2022). LNCS, vol. 13550. Springer, 2022, pp. 307-323. DOI: 10.1007/978-3-031-17108-6_19.

[9]. Ruiz-Celada O., Rosell J., Suárez R. Knowledge-Based Execution Configuration for Adaptive Behavior Trees. Journal of Intelligent & Robotic Systems, vol. 112, 2026, art. 49. DOI: 10.1007/s10846-026-02373-1.

[10]. Iovino M., Förster J., Falco P., Chung J. J., Siegwart R., Smith C. On the Programming Effort Required to Generate Behavior Trees and Finite State Machines for Robotic Applications. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 5807–5813. DOI: 10.1109/ICRA48891.2023.10160972.

[11]. Casalaro G. L., Cattivera G., Ciccozzi F., Malavolta I., Wortmann A., Pelliccione P. Model-Driven Engineering for Mobile Robotic Systems: A Systematic Mapping Study. Software and Systems Modeling, vol. 21, 2022, pp. 19-49. DOI: 10.1007/s10270-021-00908-8.

[12]. Elekes M., Molnár V., Micskei Z. Assessing the Specification of Modelling Language Semantics: A Study on UML PSSM. Software Quality Journal, 2023. DOI: 10.1007/s11219-023-09617-5.

[13]. Caldas R. D., García J. A. P., Schiopu M., Pelliccione P., Rodrigues G., Berger T. Runtime Verification and Field-Based Testing for ROS-Based Robotic Systems. IEEE Transactions on Software Engineering, vol. 50, no. 10, 2024, pp. 2544-2567. DOI: 10.1109/TSE.2024.3444697.

[14]. Dürschmid T., Timperley C. S., Garlan D., Le Goues C. ROSInfer: Statically Inferring Behavioral Component Models for ROS-Based Robotics Systems. In: Proceedings of the IEEE/ACM 46th

International Conference on Software Engineering (ICSE '24), 2024, Article 144, 13 pp. DOI: 10.1145/3597503.3639206.

[15]. Pelletier B., Lesire C., Godary-Dejean K. A Formal Framework for the Specification and Verification of Robotic Skills Composition for Autonomous Behaviors. Robotics and Autonomous Systems, vol. 192, 2025, art. 105041. DOI: 10.1016/j.robot.2025.105041.

Информация об авторах / Information about authors

Сергей Евгеньевич КОНДРАТЬЕВ является аспирантом, ассистентом кафедры автоматизированного электропривода и робототехники Липецкого государственного технического университета. Сфера научных интересов: робототехника, автоматизация, компьютерное зрение, машинное обучение.

Sergey Evgenievich KONDRATYEV – postgraduate student and assistant of the Department of Automated Electric Drive and Robotics of Lipetsk State Technical University. His research interests include robotics, automation, computer vision, and machine learning.

Николай Вадимович КАЗЮРА является магистром, преподавателем СПО кафедры автоматизированного электропривода и робототехники Липецкого государственного технического университета. Сфера научных интересов: робототехника, автоматизация, компьютерное зрение, машинное обучение.

Nikolay Vadimovich KAZYURA – master's student and junior college instructor of the Department of Automated Electric Drive and Robotics of Lipetsk State Technical University. His research interests include robotics, automation, computer vision, and machine learning.

Виктор Николаевич МЕЩЕРЯКОВ является заведующим кафедрой автоматизированного электропривода и робототехники Липецкого государственного технического университета, доктор технических наук, профессор. Сфера научных интересов: автоматизированный электропривод промышленных комплексов, мехатроника и робототехника.

Victor Nikolaevich MESHCHERYAKOV – Dr. Sci. (Tech.), Prof., head of the Department of Automated Electric Drive and Robotics of Lipetsk State Technical University. His research interests include automated electric drives for industrial complexes, mechatronics, and robotics.