(19) Евразийское патентное ведомство

(11) 042508 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2023.02.21

(21) Номер заявки
202192273

(22) Дата подачи заявки
2021.09.15

(51) Int. Cl. ***G06F 8/10*** (2006.01)
G06F 8/70 (2006.01)
G06F 11/22 (2006.01)

(54) СПОСОБ И СИСТЕМА ПРОВЕРКИ АРХИТЕКТУРЫ ПРОГРАММНО-АППАРАТНОГО РЕШЕНИЯ

(31) **2021115217**
(32) **2021.05.27**
(33) **RU**
(43) **2022.11.30**

(71)(73) Заявитель и патентовладелец:
ПУБЛИЧНОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО "СБЕРБАНК РОССИИ" (ПАО СБЕРБАНК) (RU)

(72) Изобретатель:
Панасюк Ярослав Сергеевич, Демидов Михаил Геннадьевич (RU)

(74) Представитель:
Герасин Б.В. (RU)

(56) US-A1-20060265699
US-A1-20190324744
US-A1-20200366707
US-A1-20020104071

(57) Изобретение, в общем, относится к проектированию программно-аппаратных решений, а более конкретно к автоматизированной проверке соответствия архитектуры программно-аппаратного решения архитектурным требованиям. Решаемой технической проблемой при реализации заявленного решения является обеспечение автоматизированной проверки архитектуры программно-аппаратного решения. В предпочтительном варианте реализации заявлен компьютерно-реализуемый способ проверки архитектуры программно-аппаратного решения, выполняющийся по меньшей мере одним процессором и содержащий этапы, на которых получают по меньшей мере один артефакт программно-аппаратного решения, определяют по меньшей мере одно цифровое архитектурное свойство программно-аппаратного решения, формируют реверсивную архитектуру программно-аппаратного решения, получают по меньшей мере один набор архитектурно значимых, для архитектурного контроля программно-аппаратного решения, требований и преобразуют указанный набор в единый цифровой унифицированный формат, формируют цифровой стандарт программно-аппаратного решения, выполняют проверку соответствия реверсивной архитектуры программно-аппаратного решения архитектурно значимым требованиям, содержащимся в цифровом стандарте, формируют результаты проверки программно-аппаратного решения.

042508 B1

042508 B1

Область техники

Изобретение, в общем, относится к проектированию программно-аппаратных решений, а более конкретно к автоматизированной проверке соответствия архитектуры программно-аппаратного решения архитектурным требованиям.

Уровень техники

С развитием информационных технологий IT-решения ("Information technology" - информационные технологии) стали оказывать существенное влияние на все сферы и отрасли жизнедеятельности. В настоящее время различные компании и организации активно внедряют и используют в своей структуре IT-решения. Как правило, такие решения могут быть очень сложными и должны соответствовать определенным наборам политик, правил и стандартов организации (например, требования к информационной безопасности, правила взаимодействия структурных элементов и т.д.). Для соответствия разрабатываемой информационной системы такому набору политик и стандартов, на стадии проектирования IT-решения одним из важнейших этапов является проектирование архитектуры такого программно-аппаратного решения.

Целью проектирования является определение внутренних свойств системы и детализация её внешних (видимых) свойств на основе заданных требований к программно-аппаратным средствам. Архитектура IT-решения (архитектура программно-аппаратного решения в области информационных технологий), в свою очередь, определяет выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведение в рамках взаимодействия структурных элементов, соединение выбранных элементов структуры и поведения в более крупных системах, архитектурный стиль, который направляет все элементы, их интерфейсы, их взаимодействия и их соединение. Несмотря на то, что архитектура IT-решения определяет набор структурных элементов, организованных определенными способами для взаимодействия друг с другом, складывающихся в единый программно-аппаратный комплекс, предназначенный для достижения определенных целей, существует ряд сложностей, связанных с архитектурным контролем и проверкой соответствия такого IT решения изначально заданным требованиям.

Как правило, архитектурный контроль обычно выполняется в ручном режиме на основе текстовых требований или графических рисунков и диаграмм, созданных архитекторами разрабатываемых решений.

Недостатками указанного способа могут являться неоднозначное понимание архитектурных требований на этапе разработки требований и приемки результатов, высокие трудозатраты со стороны контролирующих архитекторов на ручной контроль IT-решений, высокие трудозатраты со стороны исполнителей на процесс приемки результатов работ, повышенные трудозатраты и возможный срыв сроков проекта в случае некорректной трактовки требований исполнителями, возможность субъективной оценки IT-решений на основании текстовых документов и диаграмм, некачественный контроль и ошибки контроля из-за человеческого фактора со стороны архитекторов.

Также, из уровня техники, известен автоматизированный способ управления требованиями к разработке программного обеспечения, раскрытый в патенте США № US 8949770 B2 (LDRA TECHNOLOGY INC), опубл. 03.02.2015. Указанный способ обеспечивает возможность автоматизированной проверки соответствия архитектурных артефактов, полученных с помощью статического анализа кода, предъявляемым требованиям.

К недостаткам указанного способа можно отнести ограниченный анализ архитектурных артефактов, из-за отсутствия возможности анализа артефактов программно-аппаратного решения, не относящихся к статистическому анализу кода, таких как бинарные файлы дистрибутивов, артефакты дескрипторов развертывания, метрик и характеристик поведения информационной системы и т.д. Кроме того, указанное решение не подразумевает анализ фактически реализованных свойств IT-решения, что также ограничивает возможности архитектурного контроля IT-решения на соответствие предъявляемым архитектурным требованиям и не позволяет провести полную автоматизированную проверку всех значимых архитектурных требований. Общими недостатками существующих решений является отсутствие эффективного автоматизированного способа проверки архитектуры программно-аппаратного решения на соответствие набору архитектурных требований, обеспечивающего возможность проведения проверки с высокой точностью. Кроме того, такого рода решение должно обеспечивать возможность восстановления фактической архитектуры контролируемой информационной системы для повышения уровня автоматизации проверки, а также обеспечивать строгую формализацию архитектурных требований и их критериев контроля.

Раскрытие изобретение

Изобретение направлено на устранение недостатков, присущих существующим решениям, известным из уровня техники.

Решаемой технической проблемой в данном техническом решении является создание нового и эффективного способа проверки архитектуры программно-аппаратного решения.

Основным техническим результатом, проявляющимся при решении вышеуказанной проблемы, является обеспечение автоматизированной проверки архитектуры программно-аппаратного решения.

Дополнительным техническим результатом, проявляющимся при решении вышеуказанной пробле-

мы, является повышение точности проведения проверки архитектуры программно-аппаратного решения, за счет восстановления фактических архитектурных свойств.

Указанные технические результаты достигаются благодаря осуществлению компьютерно-реализуемого способа проверки архитектуры программно-аппаратного решения, выполняющегося по меньшей мере одним процессором и содержащего этапы, на которых:

a) получают, при помощи по меньшей мере одного анализатора, по меньшей мере один артефакт программно-аппаратного решения;

b) определяют, на основе полученного по меньшей мере одного артефакта, по меньшей мере одно цифровое архитектурное свойство программно-аппаратного решения;

c) формируют реверсивную архитектуру программно-аппаратного решения, причем, реверсивная архитектура формируется по меньшей мере на основе данных, полученных на этапе b);

d) получают по меньшей мере один набор архитектурно значимых, для архитектурного контроля программно-аппаратного решения, требований и преобразуют указанный набор в единый цифровой унифицированный формат;

e) формируют цифровой стандарт программно-аппаратного решения на основе данных, полученных на этапе d);

f) выполняют проверку соответствия реверсивной архитектуры программно-аппаратного решения архитектурно значимым требованиям, содержащимся в цифровом стандарте;

g) формируют результаты проверки программно-аппаратного решения на основе данных, полученных на этапе f).

В одном из частных примеров осуществления способа артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа бинарных файлов дистрибутивов поставки и развертывания программного обеспечения.

В другом частом варианте осуществления способа артефакт программно-аппаратного решения представляет собой по меньшей мере артефакт исходного кода решения.

В другом частном примере осуществления способа по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи лексического анализа исходного кода.

В другом частном примере осуществления способа по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи синтаксического анализа исходного кода.

В другом частном примере осуществления способа по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи семантического анализа исходного кода.

В другом частном примере осуществления способа артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа данных и метрик поведения программно-аппаратного решения. В другом частном примере осуществления способа артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа экспертных заключений архитектуры программно-аппаратного решения. В другом частном примере осуществления способа по меньшей мере один анализатор представляет собой анализатор, выбираемый из группы:

анализатор бинарных файлов дистрибутивов поставки и развертывания;

анализатор статического кода;

анализатор конфигурационных файлов;

анализатор метрик мониторинга эксплуатации;

анализатор сетевых взаимодействии;

анализатор на основе искусственного интеллекта.

В другом частном примере осуществления способа реверсивная архитектура представляет собой по меньшей мере архитектуру программно-аппаратного решения, восстановленную на основе полученных фактических артефактов.

В другом частном примере осуществления способа единый унифицированный формат представляет собой по меньшей мере программный код.

В другом частном примере осуществления способа по меньшей мере один набор архитектурно значимых требований содержит по меньшей мере один набор требований, выбираемый из группы:

требования к функциональной структуре программно-аппаратного решения;

требования к детальной архитектуре программно-аппаратного решения;

требования к информационной архитектуре программно-аппаратного решения;

требования к безопасности программно-аппаратного решения;

требования к развертываемости программно-аппаратного решения.

В другом частном примере осуществления способа требования к функциональной структуре содержат по меньшей мере данные, выбираемые из группы:

описание модулей в составе программно-аппаратного решения;

описание подмодулей программно-аппаратного решения;

описание технологических компонентов в составе программно-аппаратного решения.

В другом частном примере осуществления способа требования к детальной архитектуре содержат по меньшей мере данные, выбираемые из группы:

компонентного состава программно-аппаратного решения;

списка технологических компонент в составе программно-аппаратного решения;

взаимодействия между технологическими компонентами программно-аппаратного решения;

интеграционных взаимодействий с внешними системами.

В еще одном частном примере осуществления способа требования к информационной архитектуре содержат по меньшей мере данные, выбираемые из группы:

логической модели данных;

физической модели данных.

В другом частном примере осуществления способа требования к безопасности содержат по меньшей мере данные, выбираемые из группы:

права доступа в составе программно-аппаратного решения;

полномочия в составе программно-аппаратного решения;

ролевая модель программно-аппаратного решения;

механизмы аутентификации, используемые в программно-аппаратном решении.

В другом частном примере осуществления способа требования к развертыванию содержат по меньшей мере данные, выбираемые из группы:

диаграммы развертывания программно-аппаратного решения;

описания стендов, используемых в программно-аппаратном решении;

описания технологических ресурсов программно-аппаратного решения.

В другом частном примере осуществления способа результаты проверки отображаются пользователю.

В другом частном примере осуществления способа результаты проверки сохраняются и передаются в виде машиночитаемой структурированной информации. Кроме того, заявленные технические результаты достигаются за счет системы проверки архитектуры программно-аппаратного решения, содержащей:

по меньшей мере один процессор;

по меньшей мере одну память, соединенную с процессором, которая содержит машиночитаемые инструкции, которые при их выполнении по меньшей мере одним процессором обеспечивают выполнение способа проверки архитектуры программно-аппаратного решения.

Краткое описание фигур

Признаки и преимущества настоящего технического решения станут очевидными из приводимого ниже подробного описания и прилагаемых чертежей, на которых: фиг. 1 иллюстрирует блок-схему выполнения заявленного способа; фиг. 2 иллюстрирует систему для реализации заявленного способа.

Осуществление изобретения

Ниже будут описаны термины и понятия, необходимые для реализации настоящего технического решения.

Информационные технологии (ИТ, от англ. "Information Technology") - процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов.

Архитектура программно-аппаратного решения - организация системы, реализованная в её компонентах, их взаимоотношениях друг с другом и средой и принципах, определяющих её конструкцию и развитие.

Артефакт программно-аппаратного решения - это любой созданный искусственно элемент программной системы. Более подробно термин артефакт раскрыт в источнике информации [1].

В настоящем техническом решении под терминами: программно-аппаратное решение, IT-решение, IT-продукт, информационная система, автоматизированная система понимается совокупность программных, программно-аппаратных и/или аппаратных средств ИТ, предоставляющих определенные функциональные возможности и предназначенные для непосредственного использования или включения в различные IT-системы.

Заявленное техническое решение предлагает новый подход, обеспечивающий автоматизированную проверку архитектуры программно-аппаратного решения на соответствие предъявляемым архитектурно значимым требованиям. Основной особенностью заявленного технического решения является возможность проведения автоматизированной проверки архитектуры программно-аппаратного решения предъявляемым архитектурным требованиям за счет восстановления фактически реализованных свойств программно-аппаратного решения на основе извлеченных артефактов, что также повышает точность и качество проведения такой проверки, и, как следствие, обеспечивает возможность определения и анализа всех цифровых артефактов, содержащихся в такой архитектуре, а не только артефактов, полученных путем статистического анализа кода. Кроме того, указанное решение, также обеспечивает возможность формализации предъявляемых архитектурных требований, на основе которых выполняется проверка со-

ответствия архитектуры программно-аппаратного решения таким требованиям. Также указанное техническое решение снижает влияние человеческого фактора при реализации и контроле IT-решений.

Заявленное техническое решение может выполняться, например системой, машиночитаемым носителем, сервером и т.д. В данном техническом решении под системой подразумевается, в том числе компьютерная система, ЭВМ (электронно-вычислительная машина), ЧПУ (числовое программное управление), ПЛК (программируемый логический контроллер), компьютеризированные системы управления и любые другие устройства, способные выполнять заданную, четко определенную последовательность операций (действий, инструкций).

Под устройством обработки команд подразумевается электронный блок либо интегральная схема (микропроцессор), исполняющая машинные инструкции (программы). Устройство обработки команд считывает и выполняет машинные инструкции (программы) с одного или более устройства хранения данных, например таких устройств, как оперативно запоминающие устройства (ОЗУ) и/или постоянные запоминающие устройства (ПЗУ). В качестве ПЗУ могут выступать, но, не ограничиваясь, жесткие диски (HDD), флеш-память, твердотельные накопители (SSD), оптические носители данных (CD, DVD, BD, MD и т.п.) и др.

Программа - последовательность инструкций, предназначенных для исполнения устройством управления вычислительной машины или устройством обработки команд. Термин "инструкции", используемый в этой заявке, может относиться, в общем, к программным инструкциям или программным командам, которые написаны на заданном языке программирования для осуществления конкретной функции, такой как, например, получение артефактов программно-аппаратного решения, формирование цифрового стандарта программно-аппаратного решения, формирование результатов проверки программно-аппаратного решения, анализ данных и т.п. Инструкции могут быть осуществлены множеством способов, включающих в себя, например, объектно-ориентированные методы. Например, инструкции могут быть реализованы, посредством языка программирования C++, Java, Python, различных библиотек (например, "MFC"; Microsoft Foundation Classes) и т.д. Инструкции, осуществляющие процессы, описанные в этом решении, могут передаваться как по проводным, так и по беспроводным каналам передачи данных, например Wi-Fi, Bluetooth, USB, WLAN, LAN и т.п.

На фиг. 1 представлена блок-схема способа 100 проверки архитектуры программно-аппаратного решения, который раскрыт поэтапно более подробно ниже. Указанный способ 100 заключается в выполнении этапов, направленных на обработку различных цифровых данных. Обработка, как правило, выполняется с помощью системы, которая может представлять, например, сервер, компьютер, мобильное устройство, вычислительное устройство и т.д. Более подробно элементы системы раскрыты на фиг. 2. На этапе 110 происходит получение по меньшей мере одного артефакта программно-аппаратного решения.

На указанном этапе 110 система 200 извлекает при помощи анализаторов фактические свойства (артефакты) контролируемого программно-аппаратного решения. Артефакты программно-аппаратного решения могут представлять реальные свойства IT-решения (программно-аппаратного решения), которыми указанное решение обладает в результате разработки/сборки/сопровождения/эксплуатации. Так, в одном частном варианте осуществления, артефакты программно-аппаратного решения могут извлекаться посредством анализа программно-аппаратного решения, состоящего из по меньшей мере статического архитектурного анализа, динамического архитектурного анализа и ручного архитектурного анализа. Только статический анализ кода не позволяет узнать архитектурные свойства, касающиеся поведения работающей системы (производительность, потребление ресурсов, свойство отказоустойчивость) - как это позволяет сделать анализ артефактов поведения системы (метрики мониторинга). Экспертный анализ позволяет оцифровывать и контролировать мнения экспертов-архитекторов по поводу архитектуры системы. За счет совместного применения указанных типов анализа обеспечивается возможность анализировать и извлекать любые цифровые артефакты программно-аппаратного решения, что позволяет автоматизировать процесс проверки архитектуры на соответствие архитектурным требованиям не только в части артефактов, содержащих формальные языки программирования, но и также в части артефактов, содержащихся, например, в бинарных файлах дистрибутивов, скриптах сборки, в том числе на различных этапах DevOps-конвейера, эксплуатационных метриках, метриках мониторинга, сетевого дискаверинга и т.п. Для специалиста в данной области техники будет очевидно, что для извлечения артефактов исследуемого программно-аппаратного решения указанные типы анализа могут применяться как в комбинации, так и по отдельности.

В качестве анализаторов могут использоваться, например, анализатор бинарных файлов дистрибутивов поставки и развертывания, анализатор статического кода, анализатор конфигурационных файлов, анализатор метрик DevOps-конвейера, анализатор метрик мониторинга эксплуатации, анализатор сетевых взаимодействии, анализатор на основе искусственного интеллекта и т.д., не ограничиваясь. Стоит отметить, что термин DevOps - это сочетание культурных принципов, подходов и средств, которое улучшает способность компаний создавать приложения и сервисы на высокой скорости. С DevOps разработка и оптимизация продуктов происходит быстрее, чем при использовании традиционных процессов работы над программным обеспечением и управления инфраструктурой. Соответственно, DevOps-конвейер - это вычислительное устройство, такое как сервер, обеспечивающий набор инструментов для

реализации DevOps подхода (автоматизация процессов, отслеживание покрытия кода, инструменты взаимодействия между командами разработчиков и т.д.). Метриками DevOps, в свою очередь, могут являться такие метрики, как время цикла, частота развертывания и т.д. Указанные анализаторы широко известны из уровня техники и для специалиста в данной области техники очевидно, что может применяться любой анализатор, обеспечивающий выполнение предписанных ему функций. В одном частном варианте осуществления в качестве анализатора может быть применен парсер, обученный на поиск требуемых данных. Для специалиста в данной области техники очевидно, что парсеры и анализаторы программного обеспечения широко известны из уровня техники и могут применяться любые анализаторы, обеспечивающие выполнение предписанных им функций. Так, в одном примере реализации, анализатор может представлять собой по меньшей мере лексический анализатор кода, сконфигурированный для идентификации и лексического анализа исходного кода программно-аппаратного решения. Указанный анализатор выполнен с возможностью извлечения лексем языка программирования исходного кода. Лексемами для языка программирования могут являться, например, идентификаторы, константы, ключевые слова языка, знаки операций и разделители. Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем. Соответственно, каждой лексеме соответствует некий уникальный условный код, зависящий от типа лексемы, и дополнительная служебная информация. Такие лексемы могут быть приняты за артефакты программно-аппаратного решения.

В другом частном примере реализации анализатор может представлять собой по меньшей мере анализатор метрик мониторинга эксплуатации, предназначенный для извлечения артефактов, полученных на основе анализа данных и метрик поведения программно-аппаратного решения. Метрики мониторинга эксплуатации могут представлять, например, метрики производительности решения (число выполняемых операций в секунду), метрики потребления ресурсов - загрузка процессора, процент использования оперативной памяти, число операции ввода-вывода дисковой подсистемы. Примеры метрик более подробно описаны ниже.

Под статическим архитектурным анализом в данном решении следует понимать процесс анализа IT-решения, объектом которого являются статические цифровые артефакты такого решения. Так, например, статическими цифровыми артефактами могут являться исходный код, дистрибутивы, скрипты развертывания и т.д., не ограничиваясь. Как указывалось выше, статический архитектурный анализ может выполняться в системе 200, при помощи по меньшей мере одного анализатора, например анализатора статического кода. Указанный анализатор может быть выполнен как в виде программного средства, так и в виде программно-аппаратного средства, например анализатор статического кода может представлять вычислительное устройство, которое более подробно раскрыто на фиг. 2. В одном частном варианте осуществления статический архитектурный анализ может содержать по меньшей мере следующие этапы анализа исходного кода IT-решения: лексический анализ, синтаксический анализ, семантический анализ. На основе указанных типов анализа может быть построена модель данных статического анализа кода. Указанная модель может содержать семантическую модель, модель конфигурации, модель технологического стека, модель установки приложения, синтаксическое дерево. В данном примере реализации артефакты программно-аппаратного решения будут извлекаться из модели данных статического анализа кода.

Под динамическим архитектурным анализом в данном решении следует понимать процесс анализа IT-решения, объектом которого являются метрики динамического поведения анализируемого IT-решения в различных средах эксплуатации. Так, например, метриками динамического поведения могут являться, метрики пропускной способности, метрики утилизации ресурсов, число обрабатываемых транзакций в секунду, метрики качества обслуживания и т.д. Динамический архитектурный анализ может выполняться в системе 200, при помощи по меньшей мере одного анализатора, например анализатора метрик мониторинга эксплуатации, анализатора сетевых взаимодействий и т.д., не ограничиваясь. Указанный анализатор может быть выполнен как в виде программного средства, так и в виде программно-аппаратного средства, например анализатор метрик мониторинга эксплуатации может представлять собой вычислительное устройство, более подробно раскрытое на фиг. 2.

Под ручным архитектурным анализом в данном решении следует понимать процесс анализа IT-решения, объектом которого являются оцифрованные экспертные заключения об отдельных архитектурных характеристиках IT-решения, которые сложно или невозможно получить в полностью автоматическом режиме и требуется подключения экспертов-архитекторов. Так, например, для формализации ручного архитектурного анализа для каждого типа архитектурных характеристик, требующих экспертных заключений, может быть составлен сценарий опроса эксперта, например, в АРМ (автоматизированное рабочее место) системы 200, автоматизирующий прохождение экспертом цепочки из опросов с возможностью выбора и оценки фиксированного набора параметров данной архитектурной характеристики. Такая реализация ручного архитектурного анализа позволяет получить строго типизированную структуру данных, содержащую машинно-читаемые цифровые архитектурные характеристики исследуемого объекта.

Таким образом, на этапе 110 выполняется извлечение всех цифровых артефактов программно-аппаратного решения.

После извлечения артефактов программно-аппаратного решения система 200 переходит к выполнению этапа 120.

На этапе 120 выполняется определение, на основе полученного по меньшей мере одного артефакта, по меньшей мере одного цифрового архитектурного свойства программно-аппаратного решения.

На основе извлеченных артефактов IT-решения определяются цифровые архитектурные свойства программно-аппаратного решения. Указанный этап 120 может осуществляться, также, посредством по меньшей мере одного анализатора, описанного выше.

Рассмотрим несколько примеров определения цифрового архитектурного свойства.

Пример 1. В качестве извлеченного артефакта контролируемого программно-аппаратного решения является следующий артефакт: исходный код программной системы на Java в формате фреймворка сборки Maven. Указанный артефакт представляет собой структуру директорий и файлов с исходных кодом. Результатом определения цифрового архитектурного свойства будет являться файл java_skeleton.json, представленный ниже. Содержание файла описывает структуру программного кода системы, состоящую из набора его модулей, классов внутри этих модулей, методов и полей внутри этих классов. Все объекты в такой структуре программного кода сопровождаются атрибутами специфическими для языка программирования Java - модификаторами доступа, типами данных, аннотациями и т.д.


```
{
 "java_skeleton": {
  "artifact_id": "gac-framework",
  "group_id": "ru.sber.dka.gac",
  "version": "1.0-SNAPSHOT",
   "modules": [
    {
     "artifact_id": "gac-portal",
     "classes": [
      {
       "package": "ru.sber.gac",
       "name": "AppClass",
       "fqdn": "ru.sber.gac.AppClass",
       "imports": [
        "com.sbt.bm.ucp.api.*",
        "com.sbt.bm.ucp.entity.*",
        "com.sbt.bm.ucp.bean.*"
       ],
       "annotations": ["Data", "Service", "Bean", "Component"],
       "access_modifier": "public",
       "attributes": [
        {
         "annotations": ["NotNull", "Column"],
         "name": "param",
         "type": "String"
        }
       ],
       "methods": [
        {
         "access_modifier": "public",
         "name": "getSmartStandard",
         "parameters": [
          {
           "name": "standardId",
           "type": "Long"
          },
          {
           "name": "context",
           "type": "Object"
          }
         ],
         "return_type": "ru.sber.gac.SmartStandard"
        }
       ]
      }
     ]
    }
   ]
 }
}
```

Рассмотрим другой пример определения цифрового архитектурного свойства на основе извлеченного артефакта программно-аппаратного решения.

Пример 2. Артефакт: бинарный дистрибутив развертывания программной системы в формате Spring Boot Fat Jar. Указанный артефакт представляет собой бинарный файл. Результатом определения цифрового архитектурного свойства будет являться файл spring_boot_jar.json, представленный ниже. Содержание файла описывает структуру и содержание файла с дистрибутива развертывания, включает мета-информацию о способе создания такого файла, перечень содержащихся в нем классов, способ запуска данного файла, перечень внешних библиотек, включенных в данный файл дистрибутива.


```
{
 "spring_boot_jar": {
  "filename": "gac-portal.jar",
  "filesize": 12345438,
  "meta_inf": {
   "manifest": {
    "Manifest-Version": 1.0,
    "Created-By": "1.7.0_25 (Sun Microsystems Inc.)",
    "Main-Class": "ru.sber.dka.yspanasyuk.App"
   }
  },
  "boot_inf": {
   "classes": [
    "ru.sber.dka.gac.portal.Application",
    "ru.sber.dka.gac.portal.service.DBService"
   ],
   "lib": [
    {
     "name": "dependency1.jar",
     "artifact_id": "dependency1",
     "group_id": "sber-group-dependency",
     "version": "1.0.1"
    },
    {
     "name": "dependency4.jar",
     "artifact_id": "dependency4",
     "group_id": "sber-group-dependency",
     "version": "1.0.1"
    }
   ]
  },
  "web_inf": {
   "classes": [
    "ru.sber.dka.gac.portal.web.WebApplication",
    "ru.sber.dka.gac.portal.web.service.DBService"
   ],
   "lib": [
    {
     "name": "dependency2.jar",
     "artifact_id": "dependency2",
     "group_id": "sber-group-dependency",
     "version": "1.0.3"
    }
   ],
   "lib_provided": [
    {
     "name": "provided-dependency.jar",
     "artifact_id": "provided-dependency",
     "group_id": "sber-group-dependency",
     "version": "1.0.4"
    }
   ]
  }
 }
}
```

Таким образом, на указанном этапе 120 для каждого извлеченного артефакта определяется цифровое архитектурное свойство программно-аппаратного решения.

Далее, способ 100 переходит к этапу 130. На указанном этапе 130 формируют реверсивную архитектуру программно-аппаратного решения, на основе определенных цифровых архитектурных свойств. Реверсивная архитектура представляет собой восстановленную архитектуру IT-решения на основе фактических свойств и артефактов ИТ-решения. Реверсивная архитектура используется для контроля фактически реализованных свойств IT-решения.

Формат Реверсивной архитектуры не зависит от того, какие типы IT-решений предполагается контролировать - технологии, языки программирования, технологии сборки/установки, т.п., не ограничиваясь.

За счет формирования реверсивной архитектуры обеспечивается возможность проверки фактических цифровых артефактов IT-решения, что повышает качество и точность проверки.

На этапе 140 получают по меньшей мере один набор архитектурно значимых, для архитектурного контроля программно-аппаратного решения, требований и преобразуют указанный набор в единый цифровой унифицированный формат.

Под архитектурными требованиями в данном решении предполагается набор требований и правил (архитектурных, информационной безопасности, т.д.), которым должны удовлетворять фактические архитектурные параметры IT-решения.

Набор архитектурных требований может быть получен посредством, например, оцифровывания бумажного текста, диаграмм и т.д. После получения набора требований, указанные требования преобразуются в единый унифицированный формат. Так, например, архитектурно значимые для архитектурного контроля IT-решения требования выражаются в виде кода на одном из языков программирования и/или языков разметки (YAML, JSON, XML, и т.п.).

Архитектурно значимые требования, как правило, не имеют стандартизированного формата описания и могут излагаться, например, в виде графиков/диаграмм/блок-схем, текстовых заметок и т.д. Поэтому для строгой формализации архитектурных требований и критериев проверки, а также для снижения влияния человеческого фактора при контроле, указанный набор требований преобразуется в единый цифровой унифицированный формат.

Как указывалось выше, архитектурно значимыми требованиями является набор требований и правил, которым должно удовлетворять фактически-реализованное IT-решение. Так, к архитектурно значимым требованиям могут относиться, например, общая информация о системе - карточка автоматизированной системы, содержащая общую информацию о ней (полное/краткое название системы, критичность, статус, ФИО архитекторов и пр.). В другом частном варианте осуществления архитектурно значимые требования могут являться, например, требования к функциональной структуре программно-аппаратного решения, требования к детальной архитектуре программно-аппаратного решения, требования к информационной архитектуре программно-аппаратного решения, требования к безопасности программно-аппаратного решения и т.д. Требования к функциональной структуре, в свою очередь могут содержать по меньшей мере описание модулей в составе программно-аппаратного решения, описание подмодулей программно-аппаратного решения, описание технологических компонент в составе программно-аппаратного решения и т.д.

Требования к детальной архитектуре могут содержать по меньшей мере требования к компонентному составу программно-аппаратного решения, списку технологических компонент в составе программно-аппаратного решения, взаимодействия между технологическими компонентами программно-аппаратного решения, требования к интеграционному взаимодействию с внешними системами и т.д.

Требования к информационной архитектуре могут содержать по меньшей мере требования к логической модели данных, физической модели данных и т.д. Требования к безопасности могут содержать по меньшей мере требования к правам доступа в составе программно-аппаратного решения, полномочия в составе программно-аппаратного решения, ролевая модель программно-аппаратного решения, механизмы аутентификации, используемые в программно-аппаратном решении и т.д. Требования к развертыванию могут содержать по меньшей мере требования к диаграммам развертывания программно-аппаратного решения, описанию стендов, используемых в программно-аппаратном решении, описанию технологических ресурсов программно-аппаратного решения и т.д.

После получения архитектурно значимых, для проверки архитектуры контролируемого IT-решения, требований способ 100 переходит к этапу 150. На этапе 150 формируют цифровой стандарт программно-аппаратного решения на основе полученных архитектурно значимых требований.

Цифровой стандарт программно-аппаратного решения может представлять собой логически связанный набор требований, преобразованных в единый цифровой унифицированный формат, предназначенный для контроля IT-решений определенного класса. Цифровой стандарт может быть сформирован в машинно-читаемом/машинно-исполняемом и, одновременно, в человекочитаемом виде. Цифровой стандарт выражается в виде программного кода в декларативном или императивном виде.

Цифровой стандарт в одном частном варианте осуществления может использоваться для задания и автоматизированного выполнения действий в зависимости от выполненных результатов контроля.

Так, требования, содержащиеся в цифровом стандарте, могут использоваться в качестве части технического задания, части процедуры приемки, а также для контроля соответствия IT-решения архитектурным требованиям на ранних этапах разработки/построения IT-решения, а также для автоматизированного выполнения действий в зависимости от результатов контроля.

На этапе 160 выполняют проверку соответствия реверсивной архитектуры программно-аппаратного решения архитектурно значимым требованиям, содержащимся в цифровом стандарте. Указанный этап может быть выполнен, например, посредством модуля контроля цифрового стандарта, расположенного в системе 200. Указанный модуль выполнен с возможностью проверки фактических артефактов программно-аппаратного решения, полученных в результате восстановления реверсивной архитектуры на этапе 130, на предмет соответствия правилам/требованиям цифрового стандарта, изложенным в декларативном или императивном виде (на языках программирования Rego, Java, Python, и т.д.). Модуль контроля принимает на вход реверсивную архитектуру в виде структуры данных в одном из возможных форматов ее представления (например, JSON, YAML или XML) и проверяет ее на предмет соответствия требованиями цифрового стандарта. Сравнение с требованиями цифрового стандарта происходит посредством запуска программного кода единого цифрового унифицированного формата (в декларативном или императивном виде) и передаче ему на вход структуры данных реверсивной архитектуры. Результатом работы такого программного кода является структура данных, описывающая результат проверки требования архитектурного стандарта. Так, например, результатом проверки требований к бинарному файлу, раскрытому в примере 2, может являться сравнение фактического перечня внешних библиотек, включенных в данный файл дистрибутива, с перечнем библиотек, представленном в цифровом стандарте. Для специалиста в данной области техники будет очевидно, что каждый извлеченный фактический артефакт программно-аппаратного решения может быть проверен на соответствие предъявляемому к нему требованию аналогичным образом и данный пример не должен ограничивать варианты осуществления настоящего решения. Таким образом, за счет реализации реверсивной архитектуры обеспечивается возможность проверки реальных артефактов, созданных в процессе проектирования IT-решения, с заданным набором требований. Кроме того, как было описано выше, строгая формализация требований и их преобразование в единый цифровой унифицированный стандарт обеспечивает возможность автоматизированной проверки архитектуры исследуемого IT-решения на соответствие предъявляемым архитектурным требованиям.

Результаты проверки в модуле контроля цифрового стандарта могут быть отображены пользователю в пользовательском интерфейсе как будет описано более подробно ниже.

На этапе 170 формируют результаты проверки программно-аппаратного решения на основе данных, полученных на этапе 160.

Результатом проверки программно-аппаратного решения может являться отчет соответствия реверсивной архитектуры цифровому стандарту, который может представлять собой структуру данных, содержащую набор результатов верификации каждого из требований цифрового стандарта. Отчет о верификации может быть представлен, например, в виде структуры данных определенной схемы в формате YAML/JSON и т.д., не ограничиваясь.

Так, в одном частном варианте осуществления, результаты проверки могут быть отображены в пользовательском интерфейсе в виде таблицы, содержащей результаты проверки требований.

На основе результатов проверки система 200 может выполнять определенный набор действий, зависящих от результата проверки. Так, например, если в результате проверки реверсивная архитектура соответствует предъявляемым требованиям, система 200 может признать прохождение проверки успешной, зафиксировать факт успешного выполнения в базе данных и направить пользователю отчет об успешном прохождении проверки. Кроме того, если результаты проверки содержат только частичные результаты (например, результаты проверки одной подсистемы), то система 200 может предложить продолжить выполнение дальнейших проверок остальных правил в соответствии со сценарием или остановить проверки с информированием пользователя.

Соответственно, при неуспешной проверке архитектуры (когда реверсивная архитектура не соответствует предъявляемым требованиям), система 200 выполнена с возможностью признания прохождения проверки неуспешной, фиксации факта неуспешного выполнения в базе данных, направления пользователю отчета о неуспешном прохождении проверки. Кроме того, в одном частном варианте осуществления, при неуспешном результате проверки, пользователю может быть дополнительно отправлен список, содержащий конкретные артефакты реверсивной архитектуры, которые не соответствуют предъявляемому набору требований.

Для специалиста в данной области техники очевидно, что для отображения результатов проверки архитектуры могут быть использованы любые известные из уровня техники способы и, что данное решение не ограничивается приведенными примерами. Таким образом, в указанных материалах заявки был описан способ проверки архитектуры программно-аппаратного решения, обеспечивающий высокую точность проверки за счет восстановления реверсивной архитектуры исследуемого программно-аппаратного решения, и, как следствие высокую степень автоматизации, за счет возможности проверки любых цифровых артефактов, содержащихся в программно-аппаратном решении, предъявляемым требованиям.

На фиг. 2 представлена система (200), реализующая этапы заявленного способа (100).

В общем случае система (200) содержит такие компоненты, как: один или более процессоров (201), по меньшей мере одну память (202), средство хранения данных (203), интерфейсы ввода/вывода (504), средство В/В (505), средство сетевого взаимодействия (406), которые объединяются посредством универсальной шины.

Процессор (201) выполняет основные вычислительные операции, необходимые для обработки данных при выполнении способа (100). Процессор (201) исполняет необходимые машиночитаемые команды, содержащиеся в оперативной памяти (202). Память (202), как правило, выполнена в виде ОЗУ и содержит необходимую программную логику, обеспечивающую требуемый функционал.

Средство хранения данных (203) может выполняться в виде HDD, SSD дисков, рейд массива, флэш-памяти, оптических накопителей информации (CD, DVD, MD, Blue-Ray дисков) и т.п. Средства (203) позволяют выполнять долгосрочное хранение различного вида информации, например, истории обработки транзакционных запросов (логов), идентификаторов пользователей и т.п.

Для организации работы компонентов устройства (200) и организации работы внешних подключаемых устройств применяются различные виды интерфейсов В/В (204). Выбор соответствующих интерфейсов зависит от конкретного исполнения вычислительного устройства, которые могут представлять собой, не ограничиваясь: PCI, AGP, PS/2, IrDa, FireWire, LPT, COM, SATA, IDE, Lightning, USB (2.0, 3.0, 3.1, micro, mini, type C), TRS/Audio jack (2.5, 3.5, 6.35), HDMI, DVI, VGA, Display Port, RJ45, RS232 и т.п. Выбор интерфейсов (204) зависит от конкретного исполнения системы (200), которая может быть реализована на базе широко класса устройств, например, персональный компьютер, мейнфрейм, ноутбук, серверный кластер, тонкий клиент, смартфон, сервер и т.п.

В качестве средств В/В данных (205) может использоваться: клавиатура, джойстик, дисплей (сенсорный дисплей), монитор, сенсорный дисплей, тач-пад, манипулятор мышь, световое перо, стилус, сенсорная панель, трекбол, динамики, микрофон, средства дополненной реальности, оптические сенсоры, планшет, световые индикаторы, проектор, камера, средства биометрической идентификации (сканер сетчатки глаза, сканер отпечатков пальцев, модуль распознавания голоса) и т.п.

Средства сетевого взаимодействия (206) выбираются из устройств, обеспечивающий сетевой прием и передачу данных, например, Ethernet карту, WLAN/Wi-Fi модуль, Bluetooth модуль, BLE модуль, NFC модуль, IrDa, RFID модуль, GSM модем и т.п. С помощью средств (205) обеспечивается организация обмена данными между, например, системой (200), представленной в виде сервера и вычислительным устройством пользователя, на котором могут отображаться полученные данные (результаты проведения проверки архитектуры программно-аппаратного решения) по проводному или беспроводному каналу передачи данных, например, WAN, PAN, ЛВС (LAN), Интранет, Интернет, WLAN, WMAN или GSM.

Конкретный выбор элементов устройства (200) для реализации различных программно-аппаратных архитектурных решений может варьироваться с сохранением обеспечиваемого требуемого функционала.

Представленные материалы заявки раскрывают предпочтительные примеры реализации технического решения и не должны трактоваться как ограничивающие иные, частные примеры его воплощения, не выходящие за пределы испрашиваемой правовой охраны, которые являются очевидными для специалистов соответствующей области техники. Таким образом, объем настоящего технического решения ограничен только объемом прилагаемой формулы.

Источники информации

1. Электронный ресурс: URL:
https://en.wikipedia.org/wiki/Artifact_(software_development) (дата обращения: 15.04.2021).

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Компьютерно-реализуемый способ проверки архитектуры программно-аппаратного решения, выполняющийся по меньшей мере одним процессором и содержащий этапы, на которых:

a) получают, при помощи по меньшей мере одного анализатора, по меньшей мере один артефакт программно-аппаратного решения;

b) определяют, на основе полученного по меньшей мере одного артефакта, по меньшей мере одно цифровое архитектурное свойство программно-аппаратного решения;

c) формируют реверсивную архитектуру программно-аппаратного решения, причем реверсивная архитектура формируется по меньшей мере на основе данных, полученных на этапе b);

d) получают по меньшей мере один набор архитектурно значимых, для архитектурного контроля программно-аппаратного решения, требований и преобразуют указанный набор в единый цифровой унифицированный формат;

e) формируют цифровой стандарт программно-аппаратного решения на основе данных, полученных на этапе d);

f) выполняют проверку соответствия реверсивной архитектуры программно-аппаратного решения архитектурно значимым требованиям, содержащимся в цифровом стандарте;

g) формируют результаты проверки программно-аппаратного решения на основе данных, получен-

ных на этапе f).

2. Способ по п.1, характеризующийся тем, что артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа бинарных файлов дистрибутивов поставки и развертывания программного обеспечения.

3. Способ по п.1, характеризующийся тем, что артефакт программно-аппаратного решения представляет собой по меньшей мере артефакт исходного кода решения.

4. Способ по п.3, характеризующийся тем, что по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи лексического анализа исходного кода.

5. Способ по п.3, характеризующийся тем, что по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи синтаксического анализа исходного кода.

6. Способ по п.3, характеризующийся тем, что по меньшей мере один артефакт исходного кода программно-аппаратного решения содержит артефакт, полученный при помощи семантического анализа исходного кода.

7. Способ по п.1, характеризующийся тем, что артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа данных и метрик поведения программно-аппаратного решения.

8. Способ по п.1, характеризующийся тем, что артефакт программно-аппаратного решения представляет собой по меньшей мере один артефакт, полученный на основе анализа экспертных заключений архитектуры программно-аппаратного решения.

9. Способ по п.1, характеризующийся тем, что по меньшей мере один анализатор представляет собой анализатор, выбираемый из группы:

анализатор бинарных файлов дистрибутивов поставки и развертывания;

анализатор статического кода;

анализатор конфигурационных файлов;

анализатор метрик мониторинга эксплуатации;

анализатор сетевых взаимодействий;

анализатор на основе искусственного интеллекта.

10. Способ по п.1, характеризующийся тем, что реверсивная архитектура представляет собой по меньшей мере архитектуру программно-аппаратного решения, восстановленную на основе полученных фактических артефактов.

11. Способ по п.1, характеризующийся тем, что единый унифицированный формат представляет собой по меньшей мере программный код.

12. Способ по п.1, характеризующийся тем, что по меньшей мере один набор архитектурно значимых требований содержит по меньшей мере один набор требований, выбираемый из группы:

требования к функциональной структуре программно-аппаратного решения;

требования к детальной архитектуре программно-аппаратного решения;

требования к информационной архитектуре программно-аппаратного решения;

требования к безопасности программно-аппаратного решения;

требования к развертываемости программно-аппаратного решения.

13. Способ по п.12, характеризующийся тем, что требования к функциональной структуре содержат по меньшей мере данные, выбираемые из группы:

описание модулей в составе программно-аппаратного решения;

описание подмодулей программно-аппаратного решения;

описание технологических компонент в составе программно-аппаратного решения.

14. Способ по п.12, характеризующийся тем, что требования к детальной архитектуре содержат по меньшей мере данные, выбираемые из группы:

компонентного состава программно-аппаратного решения;

списка технологических компонент в составе программно-аппаратного решения;

взаимодействия между технологическими компонентами программно-аппаратного решения;

интеграционных взаимодействий с внешними системами.

15. Способ по п.12, характеризующийся тем, что требования к информационной архитектуре содержат по меньшей мере данные, выбираемые из группы: логической модели данных; физической модели данных.

16. Способ по п.12, характеризующийся тем, что требования к безопасности содержат по меньшей мере данные, выбираемые из группы:

права доступа в составе программно-аппаратного решения;

полномочия в составе программно-аппаратного решения;

ролевая модель программно-аппаратного решения;

механизмы аутентификации, используемые в программно-аппаратном решении.

17. Способ по п.12, характеризующийся тем, что требования к развертыванию содержат по мень-

шей мере данные, выбираемые из группы:

диаграммы развертывания программно-аппаратного решения;

описания стендов, используемых в программно-аппаратном решении;

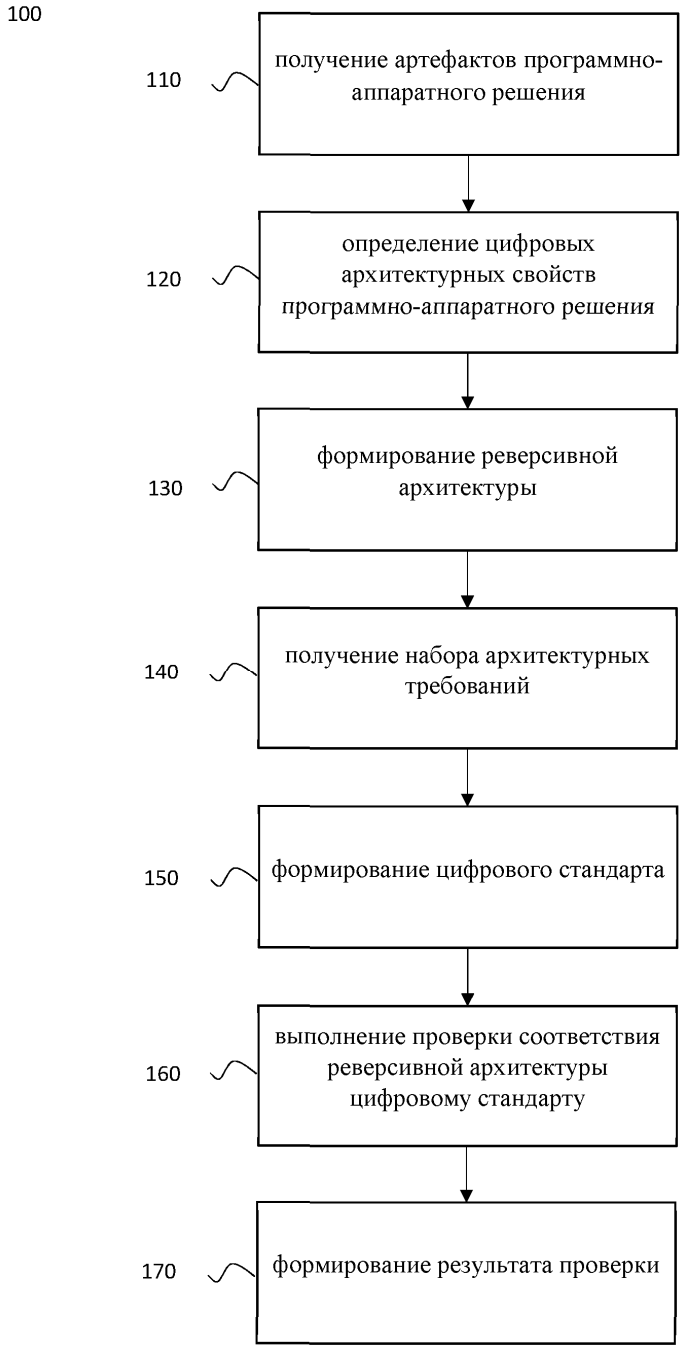
описания технологических ресурсов программно-аппаратного решения.

18. Способ по п.1, характеризующийся тем, что результаты проверки отображаются пользователю.

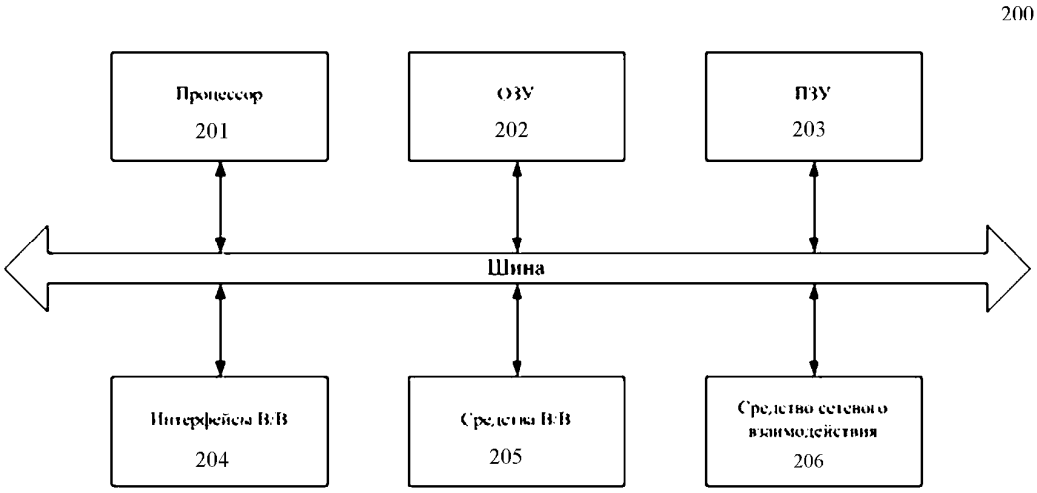
19. Система проверки архитектуры программно-аппаратного решения, содержащая:

по меньшей мере один процессор;

по меньшей мере одну память, соединенную с процессором, которая содержит машиночитаемые инструкции, которые при их выполнении по меньшей мере одним процессором обеспечивают выполнение способа по любому из пп.1-18.



Фиг. 1



Фиг. 2

Евразийская патентная организация, ЕАПВ
Россия, 109012, Москва, Малый Черкасский пер., 2