УДК 629.35-529:004.2

Поступила в редакцию 14.03.2024
Received 14.03.2024

И. Н. Носырев[1], А. Г. Савчиц[2], М. М. Татур[3], Д. Ю. Перцев[3]

[1] ГНУ «Объединенный институт машиностроения НАН Беларуси»
Минск, Республика Беларусь
E-mail: nosureviluha@mail.ru
[2] ОАО «МАЗ» – управляющая компания холдинга «БЕЛАВТОМАЗ»
Минск, Республика Беларусь
E-mail: savchits@maz.by
[3] Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
E-mail: tatur@bsuir.by, pertsev@bsuir.by

ОТКРЫТАЯ АРХИТЕКТУРА СИСТЕМЫ УПРАВЛЕНИЯ МОБИЛЬНОЙ РОБОТИЗИРОВАННОЙ ПЛАТФОРМЫ: ОТ ОНТОЛОГИИ К ТЕХНИЧЕСКОМУ РЕШЕНИЮ

Аннотация. Рассмотрены подходы к проектированию и построению открытых архитектур, проблемы, возникающие при выполнении системного проектирования. Предложена распределенная система управления грузового электромобиля как элемент открытой архитектуры.

Ключевые слова: открытая архитектура, система управления грузового транспорта, системное проектирование.

I. N. Nosarau[1], A. G. Savchits[2], M. M. Tatur[3], D. Yu. Pertsau[3]

[1] SSI "The Joint Institute of Mechanical Engineering
of the National Academy of Sciences of Belarus"
Minsk, Belarus
E-mail: nosureviluha@mail.ru
[2] OJSC "MAZ – the Management Company of the BELAVTOMAZ Holding"
Minsk, Belarus
E-mail: savchits@maz.by
[3] Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
E-mail: tatur@bsuir.by, pertsev@bsuir.by

OPEN ARCHITECTURE OF THE CONTROL SYSTEM OF A MOBILE ROBOTIC PLATFORM: FROM ONTOLOGY TO TECHNICAL SOLUTION

Abstract. Approaches to the design and construction of open architectures and problems that arise when performing system design are considered. A distributed control system for an electric cargo vehicle is proposed as an element of an open architecture.

Keywords: open architecture, freight transport control system, system design.

Введение

Прежде чем говорить об «открытой архитектуре», давайте уточним понятие «архитектура», с которого и начнем исследование. Архитектура вычислительной машины (Computer architecture) в ГОСТ 15971-90 определена как концептуальная структура, определяющая проведение обработки информации и принципы взаимодействия технических средств и программного обеспечения. Таким образом, в настоящей работе будем считать, что архитектура – наиболее общая категория вычислительной системы, включающая (или предопределяющая) функциональное, структурное, информационное, алгоритмическое, аппаратное и программное содержание.

Разработка архитектуры вычислительной системы является верхним, самым ответственным этапом проектирования и выполняется наиболее квалифицированными специалистами – системными архитекторами. Общепринятой методологией системного проектирования выступает разработка спецификации функций (функционала в общем смысле или функциональной схемы в частности) с последующей структурной, алгоритмической, а затем – программно-аппаратной реализацией (рис. 1) [1]. Именно на этапе системного проектирования от того, как задумана и спроектирована архитектура, в основном определяются технико-эксплуатационные, экономические характеристики и ограничения будущего технического решения.

Архитектура может быть формально описана в виде графической мнемоники, алгоритмически или даже программным кодом. Например, известны типовые архитектуры персональных компьютеров семейства x86, микроконтроллеров или определенного семейства FPGA, архитектура конкретной нейросети, клиент-серверная архитектура, архитектура конкретного программного комплекса и т. п.

Под открытой архитектурой понимают формальное представление программно-аппаратного вычислительного комплекса, позволяющее добавлять и/или модифицировать функционал системы. Свойство открытости архитектуры обычно связывают с использованием технических решений с открытым стандартом либо опубликованным описанием. Так, компьютеры с открытой аппаратной архитектурой могут содержать множество стандартизованных слотов расширения, позволяющих сторонним производителям оборудования создавать устройства, а пользователям свободно устанавливать их, создавая оригинальные конфигурации. Аналогично, открытая программная архитектура позволяет добавлять дополнительные программные модули к базовому ПО или модифицировать существующие. Открытые API (Application Program Interface) являются инструментом изменения или расширения базовой функциональности программных продуктов. Также программная архитектура может считаться открытой, когда обмен сообщениями между программными модулями имеет стандартную структуру, формат данных и т. п.

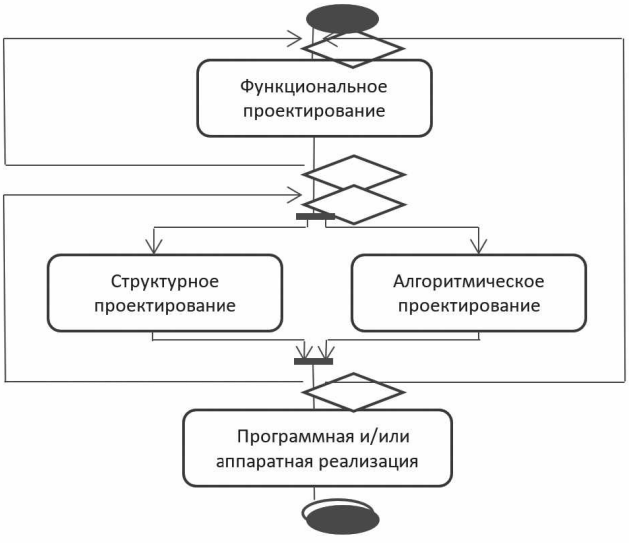


Рис. 1. Взаимосвязь этапов системного проектирования и программно-аппаратной реализации вычислительного комплекса

1. Примеры открытых архитектур глобального уровня

Одна из первых версий архитектуры *Future Avionics Capability Environment (FACE)* [2] была спроектирована в 2010 г. и поддерживается консорциумом The Open Group. Архитектура предназначена для разработки и сопровождения программного обеспечения реального времени для авионики всех типов военных воздушных платформ. Сегодня в Open Group входят поставщики, заказчики, научные круги и пользователи.

Эталонная архитектура FACE состоит из 5 логических сегментов (рис. 2), функционал и интерфейс взаимодействия между которыми строго регламентирован:

– сегмент операционной системы (OSS);

– сегмент служб ввода/вывода (IOSS), используемый для предоставления единого интерфейса для получения информации от аппаратного обеспечения, выпускаемого различными производителями;

– сегмент служб, специфичных для платформы (PSSS), включает сервисы жизнеобеспечения и конфигурации, которые применяются почти в каждой летающей системе;

– сегмент транспортных служб (TSS), на котором определен стандарт взаимодействия (например, DDS [3], CORBA [4]), которые обеспечивают обмен информацией, не зависящий от местоположения, аппаратной платформы, языка программирования или операционной системы (например, применение протоколов TCP/IP, UDP/IP, QoS, SSL/TLS и т. п.);

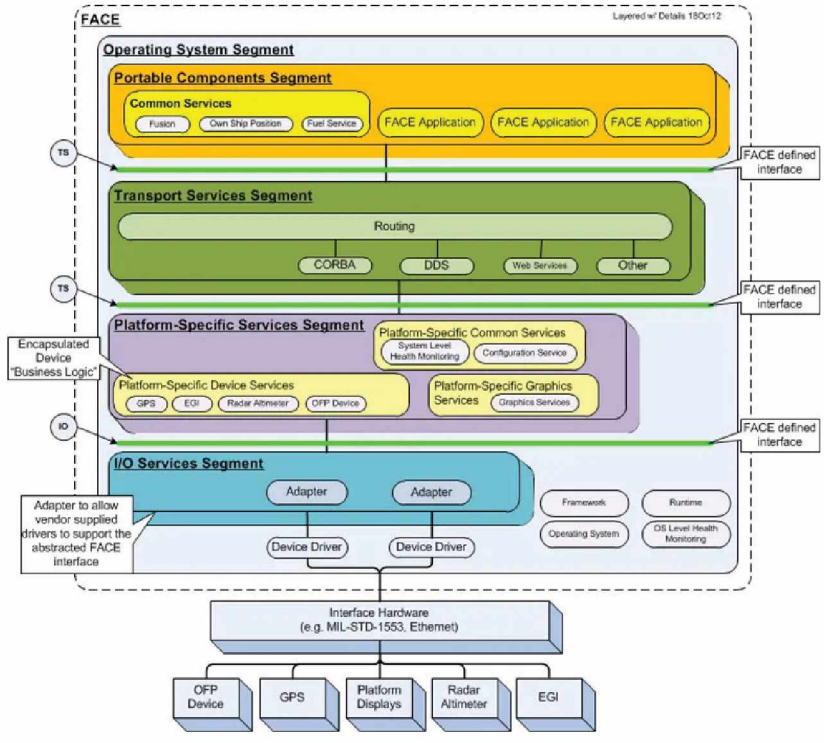


Рис. 2. Эталонная архитектура FACE

– сегмент переносных компонентов (PCS), который определяет прикладное программное обеспечение (например, управление положением самолета, управление топливом или радиолокационной информацией).

Для сегмента операционной системы определены три профиля, в которых адаптируются интерфейсы прикладного программирования (API) операционной системы, языки программирования, возможности языков программирования, режимы выполнения, фреймворки и графические возможности для удовлетворения требований к программным компонентам различного уровня критичности (т. е. по сути определяют уровень безопасности, которому должна удовлетворять разрабатываемая архитектура):

– Security, который ограничивает API-интерфейсы ОС минимальным набором полезных функций, позволяя оценивать функции безопасности с высокой степенью надежности, выполняемые как отдельный процесс;

– Safety, который ограничивает API ОС теми, которые имеют сертификацию безопасности;

– General Purpose, который поддерживает API ОС, отвечающие детерминированным требованиям реального времени или недетерминированным требованиям нереального времени, в зависимости от реализации системы или подсистемы.

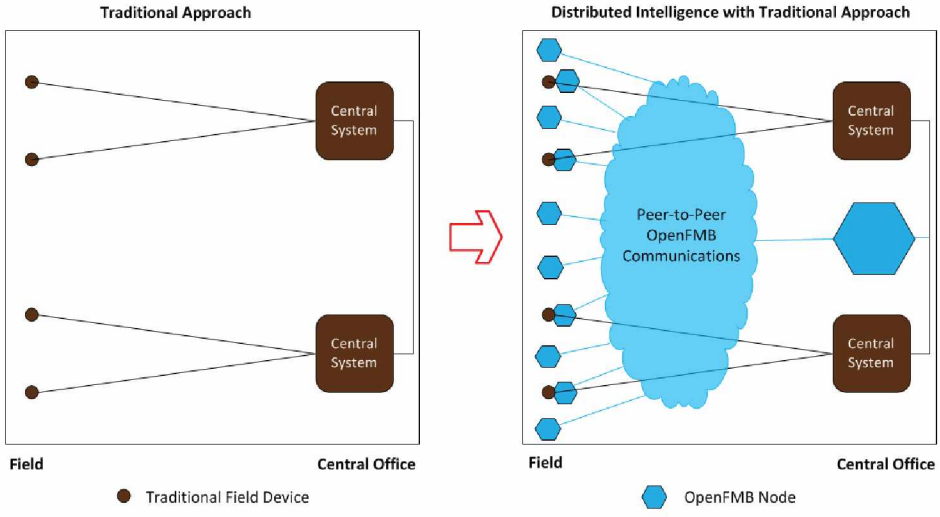


Рис. 3. Эталонная архитектура Open FMB

Open Field Message Bus (OpenFMB) [5] является развитием протоколов взаимодействия с системами мониторинга, установленными на конечных устройствах (например, на линиях электропередач, трансформаторных подстанциях, жилых домах), независимо от производителя оборудования, и связана с поэтапным переходом на 5G сети и базовой обработкой информации на местах (так называемый Edge Computing).

OpenFMB – это эталонная архитектура для взаимодействия с общей моделью данных, основанной на стандартах IEC 61968/61970 (CIM) и IEC 61850 (рис. 3). Разработана коалицией и определена в UML с помощью Enterprise Architect, что обеспечивает независимость от языка программирования и технологии. В коалицию входят лидеры отрасли, национальные лаборатории, независимые поставщики программного и аппаратного обеспечения.

Архитектура обеспечивает одноранговую связь в гетерогенной экосистеме поставщиков, устройств и программного обеспечения (рис. 4). В практических реализациях взаимодействие осуществляется с помощью XMI или языка определения интерфейсов (IDL) с использованием подхода платформонезависимого моделирования.

AUTOmotive Open System ARchitecture (AUTOSAR) [6] возникло в 2003 году в результате партнерства заинтересованных в развитии автомобилестроения сторон (Bavarian Motor Works (BMW), Robert Bosch GmbH, Continental AG, Daimler AG, Siemens VDO и Volkswagen). Целью является создание и внедрение открытой и стандартизированной архитектуры программного обеспечения для автомобильных электронных блоков управления. Спецификация описывает подходы к масштабируемости на различные варианты

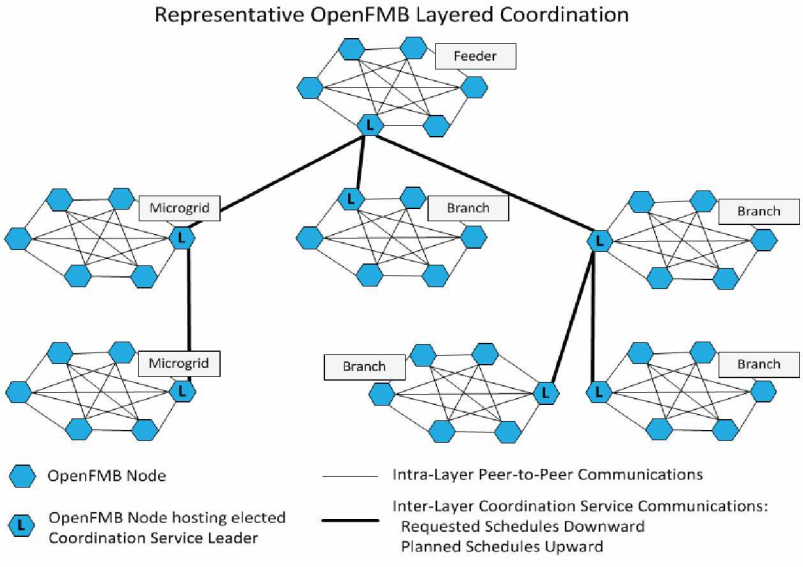


Рис. 4. Многоуровневое взаимодействие в архитектуре OpenFMB

транспортных средств и платформ, переносимость программного обеспечения, учет требований доступности и безопасности.

Спецификация включает в себя описание базовых программных модулей, определяет интерфейсы приложений и формирует общую методологию разработки на основе стандартизированного формата обмена. Базовые программные модули, доступные благодаря многоуровневой программной архитектуре AUTOSAR, могут использоваться в автомобилях разных производителей и электронных компонентах разных поставщиков, что позволяет сократить расходы на исследования и разработки.

AUTOSAR использует трехслойную архитектуру (рис. 5):

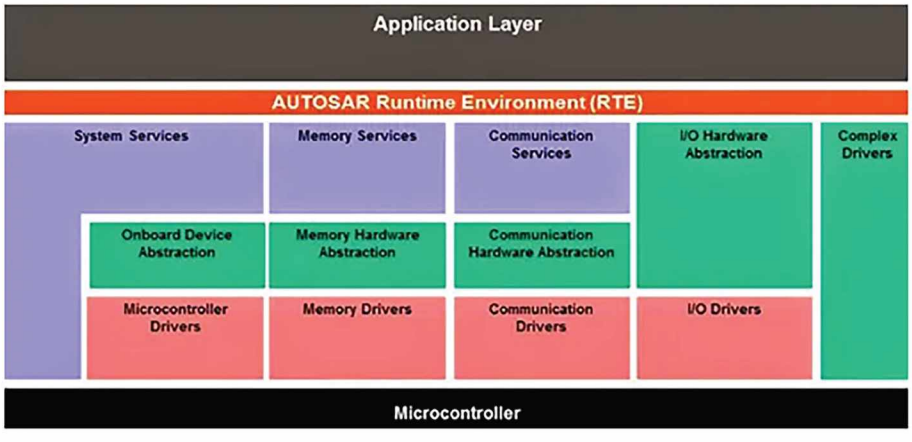


Рис. 5. Архитектура AUTOSAR

– базовое программное обеспечение: стандартизированные программные модули, не имеющие явного автомобильного назначения, но предоставляющие услуги, необходимые для запуска функциональной части верхнего программного слоя. Включает три уровня: услуги (управление обновлениями и конфигурациями, управление состоянием, управление сетью, диагностика), абстракцию электронного блока управления (ECU), абстракцию микроконтроллера;

– среда выполнения (RTE): промежуточное программное обеспечение, абстрагирующееся от топологии сети для обмена информацией между компонентами прикладного программного обеспечения и между базовым программным обеспечением и приложениями. Состоит из множества сервисов, разделенных на функциональные группы и представляющие инфраструктуру для системных, запоминающих и коммуникационных блоков;

– прикладной уровень: компоненты прикладного программного обеспечения, которые взаимодействуют со средой выполнения.

Как видно из приведенных примеров открытых архитектур, единого стиля, проформы, дизайна и правил их представления нет, однако общим является стремление отразить основные, концептуальные информационные и структурные принципы построения соответствующих информационных (вычислительных) систем.

2. Открытые архитектуры в контексте системного проектирования вычислительных систем

Как следует из рис. 1, системное проектирование включает функциональное, структурное и алгоритмическое проектирование. Функционал системы представляется в виде набора взаимодействующих между собой функций (или функциональных блоков, модулей), а каждая из функций должна быть раскрыта или реализована алгоритмически. Для обеспечения открытости архитектуры на системном уровне известны два альтернативных подхода.

Первый – централизованный (монолитный). В данном случае для обеспечения взаимодействия функциональных блоков создается супервизор (ядро), которое несет в себе глобальный функционал системы и обеспечивает совместную работу всех функциональных блоков для достижения общей цели. Доступ к функционалу осуществляется с помощью единого API (на рис. 6 выделен желтым). Тогда в случае добавления новых функций или модификации системных связей между функциями необходимо корректировать ядро системы, т. е. вносить существенные изменения в систему, которые при этом могут быть скрыты от пользователей API.

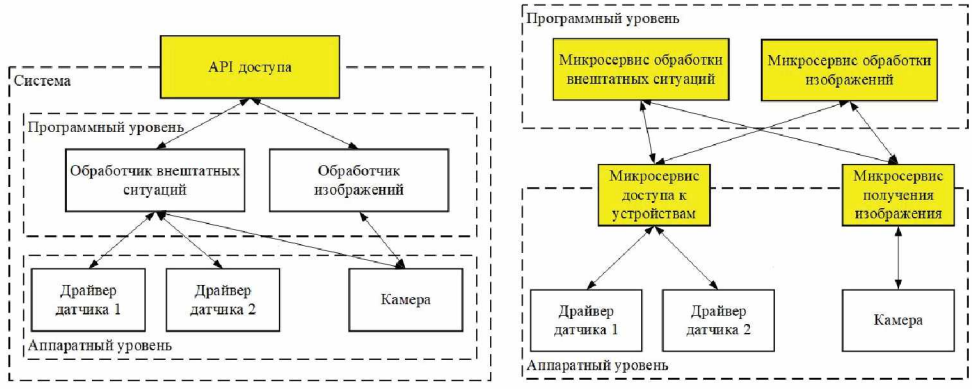


Рис. 6. Альтернативные подходы для обеспечения открытости архитектур:
а – централизованный (монолитный); *б* – децентрализованный (микросервисный)

Второй – децентрализованный (микросервисный). В данном случае функциональные блоки представляются как достаточно самостоятельные сущности (агенты), взаимодействующие с внешней средой, а общий результат работы системы определяется не общим детерминированным алгоритмом, а эффектом от взаимодействия агентов со средой (рис. 6). Подобные системы иногда называют агентными. При этом с одной стороны все агенты (сервисы) могут взаимодействовать между собой, а с другой – внешняя среда может взаимодействовать с сервисами, к которым был открыт доступ извне. Их отличительные свойства – адаптивность (интеллектуальность) к внешним условиям, возможность работать в динамической неструктурированной среде. Например, агентный подход может быть применен при системном проектировании нейросетей с функцией обучения с подкреплением, семантических сетей, системы управления и безопасности робототехнических комплексов и др.

Как для централизованного, так и для децентрализованного подходов к организации архитектуры добавление и/или модификация функционала может осуществляться следующими способами: масштабированием, заменой реализации функции и опциональным добавлением новой функции.

Схематично названные способы наращивания и модификации функционала системы поясним на примере рис. 7. Пусть система состоит из четырех функциональных блоков, Ф1 – функция технического зрения, Ф2 – функция внешнего управления, Ф3 – функция автопилота, Ф4 – функция позиционирования (рис. 7, *а*).

Масштабирование – «горизонтальное» наращивание имеющихся компонентов (программных и/или аппаратных), аналогичных или идентичных к уже действующим. При этом архитектура должна поддерживать такое наращивание как программно (в т. ч. информационно, алгоритмически),

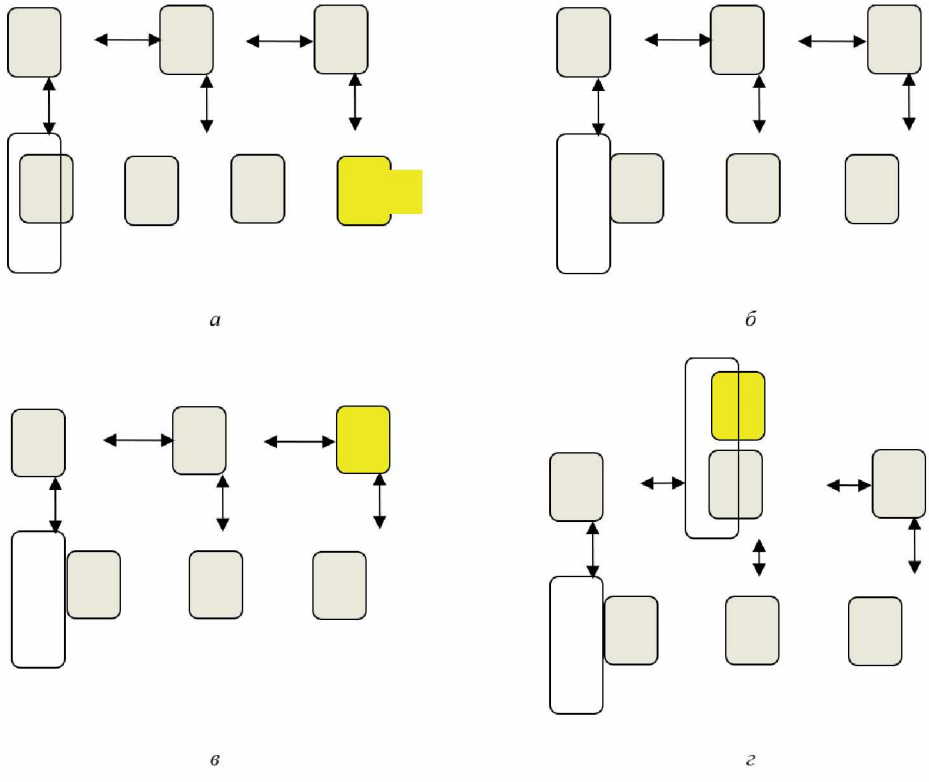


Рис. 7. Способы наращивания и модификации функционала системы:
а – исходный функционал; *б* – масштабирование функции; *в* – замена функции;
г – добавление функции

так и аппаратно (в т. ч. конструктивно). Например, если функция технического зрения Ф1 обеспечивает получение и обработку изображений от 3 камер, тогда масштабирование функции Ф1 будет состоять в дополнении еще одной или нескольких камер (рис. 7, *б*).

Замена реализации функции – удаление действующей и внесение на ее место новой. При этом может меняться алгоритмическая, программная и аппаратная реализация функции, технические параметры (точность, быстродействие и т. п.), однако информационно и конструктивно замена функции должна вписываться в действующую архитектуру. Например, если функция позиционирования мобильной платформы Ф4 обеспечивает определение координат в заданной системе, возможно, с учетом информации с подсистемы технического зрения, тогда замена функции Ф4 на функцию Ф4* с полным сохранением информационного (электрического, конструктивного) интерфейса будет для системы «незаметной, скрытой», при этом технические, эксплуатационные, экономические параметры системы могут быть улучшены (рис. 7, *в*).

Добавление функции – расширение функциональных возможностей системы за счет опционального добавления новой функции в дополнение к действующей. При этом архитектура должна поддерживать такое дополнение, а объем системных доработок, связанных с дополнением новой функции, характеризует уровень открытости архитектуры. Например, если функция автопилота Ф3 предусматривает некоторый (один) вариант управления, тогда возможно добавление альтернативной функции управления Ф3*, которая будет включаться в работу в зависимости от внешних условий (рис. 7, *г*).

Несложно заметить, что приведенные способы, которыми разработчик может менять функционал системы, различаются по гибкости. Так, замена реализации функции (*в*) не требует системных изменений, а масштабирование (*б*) и добавление функций (*г*) могут быть связаны с системными доработками. То есть понятие открытости архитектуры относительно, а степень свободы в модификации функционала системы определяется (или ограничена) архитектурными особенностями системы.

3. Открытые архитектуры в контексте программно-аппаратной реализации вычислительных систем

В общем виде система может быть представлена композицией устройств (компонент) и интерфейсов, которые, в свою очередь, могут быть реализованы как программно, так и аппаратно (рис. 8).

Системный уровень описывает порядок и правила обеспечения взаимодействия отдельных подсистем (устройств, компонентов) и интерфейсов, направленных на выполнение всех функций, предусмотренных конструкцией транспортного средства. В качестве примера рассмотрим распределенную архитектуру системы управления бортовой электроникой грузового электромобиля МАЗ 4381ЕЕ (рис. 9). Эта архитектура с определенными модификациями может быть выбрана за основу для построения системы управления роботизированной мобильной платформы на базе МАЗ.

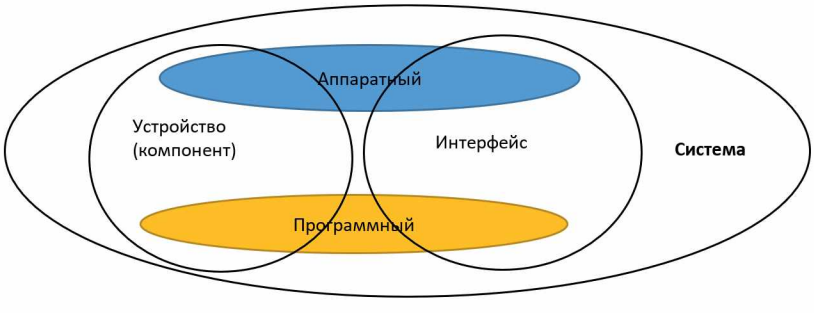


Рис. 8. Мнемосхема уровней представления (описания) системы

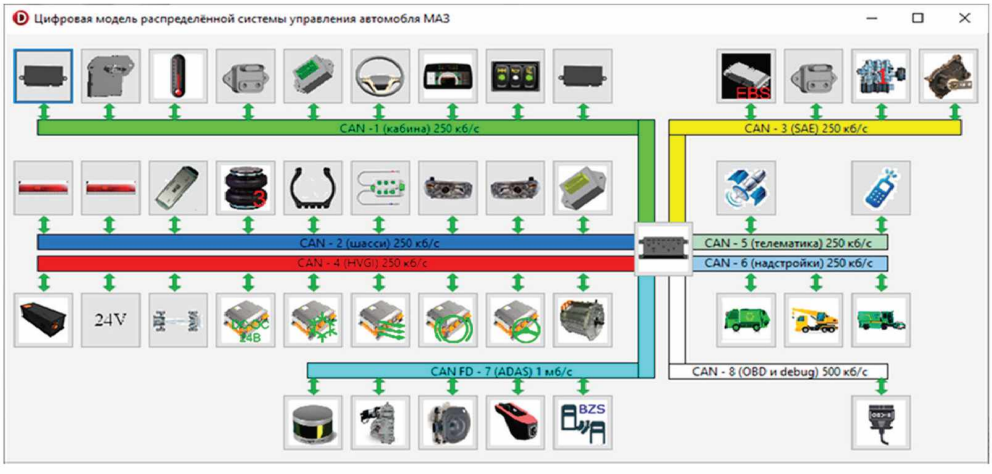
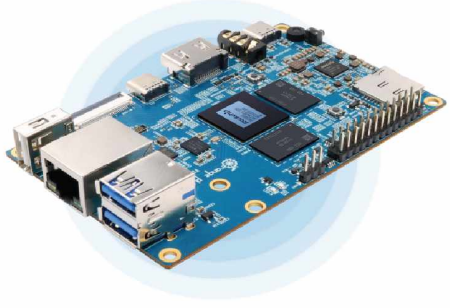


Рис. 9. Структурная схема распределенной системы управления грузового электромобиля



CPU8-core 64 core architecture, 4*Cortex-A76 + 4*Cortex-A55

GPU ARM Mali-G610

NPU 6Tops AI computing power

Support INT4/INT8/INT16 hybrid computing, empowering various AI scenarios

1000Mbps Ethernet

Interfaces: UART, PWM, I2C, SPI, CAN and GPIO.

Supported OS: Ubuntu + ROS

Рис. 10. Внешний вид платы Orange Pi5 с NPU с описанием

В качестве компонентов системы выступают подсистемы управления электродвигателем, батареей, тормозами, а также различные электроприводы, датчики, светотехника и др. Структурирование и взаимодействие подсистем и компонентов обеспечивается за счет интерфейсов. Наиболее распространенными интерфейсами на транспортных средствах являются CAN-bus, LIN, FlexRay, Ethernet. Указанные интерфейсы должны соответствовать ряду критериев как на физическом, так и на программном уровнях, что обеспечивает подключение различных периферийных устройств, удовлетворяющих принятым стандартам обмена. В качестве центрального процессора может быть использован любой встраиваемый компьютер с универсальной архитектурой и операционной системой реального времени, вычислительная мощность которого обеспечит реализацию алгоритмов управления, а интерфейсы – связь с компонентами. Например, одноплатный компьютер Orange Pi5 с NPU (рис. 10).

Для аппаратной реализации компонентов и интерфейсов применяются микроконтроллеры и прочие как активные, так и пассивные электронные приборы класса Automotive. Это особая группа элементной базы с гарантированным высоким уровнем надежности с жесткими эксплуатационными требованиями по температуре, вибрации, пыле- и влагозащищенности, электромагнитной совместимости и пр. Кроме того, в этом классе существуют компоненты категорий ASIL (Automotive Safety Integrity Level) с различными уровнями полноты автомобильной безопасности. ASIL – это схема уровней рисков, определенная стандартом ГОСТ Р 26262 (ISO 26262) [7], которая помогает установить требования безопасности путем анализа сценариев потери управляемости транспортного средства и возможных последствий.

Открытая архитектура программного обеспечения должна предусматривать взаимодействие всех компонентов программного обеспечения (методов, подпрограмм-функций) в общей структуре и содержать описание интерфейсов между всеми компонентами программного обеспечения, последовательность выполнения процессов, зависимость выполнения процессов в реальном времени. Наиболее распространенными платформами, на которых строятся системы управления робототехнических комплексов, являются:

– MSRDS10, Microsoft Robotics Developer Studio, Microsoft, U.S.;

– ERSP11, Evolution Robotics Software Platform, Evolution Robotics, Europe;

– ROS, Robot Operating System, Open Robotics12, U.S. (далее ROS);

– OpenRTM, National Institute of Adv. Industrial Science and Technology (AIST), Japan;

– OROCOS, Europe;

– OPRoS, ETRI, KIST, KITECH, Kangwon National University, South Korea.

Наиболее популярным из представленных вариантов и неофициальным стандартом в робототехнических системах с открытым исходным кодом является ROS. Она имеет наибольшее сообщество разработчиков, широкую линейку уже разработанных модулей и достаточно подробную документацию.

В целом открытая архитектура программного обеспечения обеспечивает:

– возможность разработки и модификации отдельных программных модулей;

– конфигурацию программного обеспечения (внесение доработок верхнего, системного уровня);

– тестируемость программного обеспечения на всех уровнях;

– удобство сопровождения проекта архитектуры программного обеспечения.

Заключение

Система управления мобильной роботизированной платформы является сложным по своей сути объектом проектирования по следующим причинам.

Механическая часть (несущее шасси) может иметь различные способы привода, движителей, руления, различные массогабаритные характеристики, различные агрегаты навесного оборудования, что в целом обуславливает ее целевое применение. Поскольку эта сфера машиностроения только начинает развиваться, то требования по унификации и стандартизации мобильных роботизированных платформ еще не сформировались. Как следствие, при разработке системы управления мобильной платформой необходимо в полном объеме выполнять полный цикл системного (функционального, структурного и алгоритмического проектирования). Очевидно, это связано с высокой трудоемкостью и затратами на проектирование. Одним из подходов для повышения эффективности разработки на данном этапе является модельно-ориентированное проектирование [8, 9].

Проведенный анализ существующих открытых архитектур, практический опыт реализации отдельных мобильных роботизированных платформ позволяет сформулировать некоторые рекомендации по выбору архитектуры программно-аппаратного обеспечения системы управления, составу оборудования, используемым интерфейсам.

Мехатроника является «связующим звеном» механической части мобильного робота с программно-аппаратным обеспечением мобильной платформы. В большинстве случаев датчики и исполнительные устройства стандартизированы по электрическим характеристикам и информационным протоколам. Современные автомобили и тем более мобильные роботизированные платформы имеют сотни мехатронных компонентов, управление которыми невозможно без применения бортовых сетевых коммутационных интерфейсов. В машиностроении приоритет отдан CAN-шине, посредством которой обеспечивается распределенное управление периферийными устройствами [10].

Вычислительным ядром системы управления (по крайней мере, для этапа прототипирования) может стать встраиваемый компьютер с операционной системой Linux (Ubuntu) под управлением фреймворка ROS. Это позволит с небольшими трудозатратами осуществлять подключение сенсоров системы навигации (GPS-приемников, камер, лидаров, радаров, сонаров), а также в реальном времени управлять периферийными устройствами посредством автопилота. Необходимым компонентом любой мобильной роботизированной платформы являются средства связи, обеспечивающие удаленный доступ к роботу для мониторинга и директивного управления.

Список использованных источников

1. Новиков, Ф. А. Моделирование на UML / Ф. А. Новиков, Д. Ю. Иванов. – СПб. : «Профлит», 2010. – 640 с.

2. Open Group FACE® Consortium [Электронный ресурс]. – Режим доступа: https://www.opengroup.org/face. – Дата доступа: 01.03.2024.

3. Data Distribution Service [Электронный ресурс]. – Режим доступа: http://www.omg.org/spec/DDS. – Дата доступа: 01.03.2024.

4. Common Object Request Broker Architecture [Электронный ресурс]. – Режим доступа: http://www.corba.org. – Дата доступа: 01.03.2024.

5. OpenFMB.io [Электронный ресурс]. – Режим доступа: https://openfmb.gitlab.io. – Дата доступа: 01.03.2024.

6. Automotive Open System Architecture [Электронный ресурс]. – Режим доступа: https://www.autosar.org. – Дата доступа: 01.03.2024.

7. What is the ISO 26262 Functional Safety Standard? [Электронный ресурс]. – Режим доступа: http://www.ni.com/white-paper/13647/en/#toc2. – Дата доступа: 01.03.2024.

8. ЦИТМ Экспонента [Электронный ресурс]. – Режим доступа: www.exponenta.ru. – Дата доступа: 01.03.2024.

9. Татур, М. М. Методика модельно-ориентированного проектирования алгоритмов управления мобильными роботами = Methodology for Model-Based Design of Mobile Robots Control Algorithms / М. М. Татур, Н. С. Игнатюк, А. Д. Коников // Доклады БГУИР. – 2024. – Т. 22, № 1. – С. 91–99.

10. Распределенная система управления бортовыми электронными устройствами: разработка и внедрение инновационной технологии / С. Н. Поддубко [и др.] // Новости науки и технологий. – 2023. – № 1 (64). – С. 14–24.