УДК 004.7 DOI:10.30724/1998-9903-2022-24-3-175-184

МЕТОДЫ ДИНАМИЧЕСКОГО ПЕРЕПРОГРАММИРОВАНИЯ ДЛЯ МОБИЛЬНОГО РОБОТА С МОДУЛЬНОЙ АРХИТЕКТУРОЙ

Андреев[1] В.П., Плетенев[1,2] П.Ф.

Московский государственный технологический университет «СТАНКИН», г. Москва, Россия
[2]АО «ВНИИЖТ», г. Москва, Россия
andreevipa@yandex.ru
ORCID: https://orcid.org/0000-0001-6286-112X, cpp.create@gmail.com*

Резюме: *ЦЕЛЬ. Анализ существующих и перспективных методов динамического перепрограммирования, пригодных для использования в мобильном роботе с модульной архитектурой системы управления (СУ). Кратко описана функционально-модульная архитектура системы управления мобильного робота, реализующая распределённые вычисления, что позволяет обеспечить режим работы СУ в реальном времени при использовании в СУ модулей встраиваемых систем – микроконтроллеров невысокой производительности. Рассмотрены особенности применения 4-х методов по 6 критериям: применимость на разных встраиваемых микропроцессорных системах, объём требуемой оперативной памяти и памяти программ, скорость вычислений, теоретическая сложность создания реализации метода, теоретическая сложность использования метода конечным пользователем (настройщиком), гибкость создаваемого метода. В качестве результата исследования приведены рекомендации по применению рассмотренных методов.*

Ключевые слова: *мобильный робот; модульный робот; система управления; коммуникационный канал; динамическое перепрограммирование; встраиваемые системы.*

Благодарности: *Результаты получены в рамках работ по гранту РФФИ № 19-07-00892а.*

Для цитирования: Андреев В.П.,. Плетенев П.Ф. Методы динамического перепрограммирования для мобильного робота с модульной архитектурой // Известия высших учебных заведений. проблемы энергетики. 2022. Т.24. № 3. С. 175-184. doi:10.30724/1998-9903-2022-24-3-175-184.

DYNAMIC REPROGRAMMING METHODS FOR A MOBILE ROBOT WITH MODULAR ARCHITECTURE

VP. Andreev[1], PF. Pletenev[1,2]

[1]Moscow State University of Technology «STANKIN», Moscow, Russia
[2]AO «VNIIZhT», Moscow, Russia
andreevipa@yandex.ru
ORCID: https://orcid.org/0000-0001-6286-112X, cpp.create@gmail.com*

Abstract: *THE PURPOSE of this article is to analyze existing and promising methods of dynamic reprogramming suitable for use in a mobile robot with a modular control system architecture. The article briefly describes functional-modular architecture of the control system (CS) of a mobile robot implementing distributed computing, which makes it possible to ensure the real-time operation of the modules' CS even when it's built using low-power embedded systems. The article describes features of 4 different methods according to 6 criteria: applicability on different embedded systems, the amount of operating and program memory required, the speed of calculations, the theoretical complexity of creating an implementation of the method, the theoretical complexity of using the method by the end user (tuner), the flexibility of the method being created. The results of this study are given in form of recommendations for the application of the considered methods in different environments and purposes.*

Keywords: *мmobile robot; modular robot; control system; communications channel; dynamic reprogramming; embedded systems.*

Acknowledgments: *Research was supported by the Russian Foundation for Basic Research: Grant 19-07-00892a.*

For citation: Andreev VP, Pletenev PF. Dynamic reprogramming methods for a mobile robot with modular architecture. *Power engineering: research, equipment, technology.* 2022;24(3): 175-184. doi:10.30724/1998-9903-2022-24-3-175-184.

Введение

Опыт применения робототехнических комплексов (РТК) в экстремальных условиях (авария на чернобыльской АЭС в 1986 г.) показал, что для точного соответствия функционала РТК поставленной задаче при работе в агрессивных для человека средах (космос, зоны химического или радиоактивного заражения) требуются не только роботы в специальном исполнении (экстремальная робототехника), но может потребоваться ***оперативное*** изменение состава и/или структуры РТК непосредственно на месте проведения работ. На необходимость данного функционала для экстремальной робототехники указывали такие известные отечественные учёные, как академик РАН д.т.н. Попов Евгений Павлович [1], д.т.н. Юревич Евгений Иванович [2] и д.ф.-м.н. Платонов Александр Константинович [3]. Иными словами, для таких применений необходимы реконфигурируемые роботы.

Реконфигурация – изменение конфигурации модульного робота для достижения запланированного изменения функции модульного робота[1].

Фактически реконфигурация – это целенаправленное изменение конфигурации и/или технических характеристик мобильного робота (МР) при изменении состава его функциональных узлов без необходимости перенастройки программного обеспечения его системы управления (СУ). Но подобная реконфигурация возможна лишь при модульном решении архитектуры системы управления РТК. Пример такой архитектуры приведён на рис.1 [4].

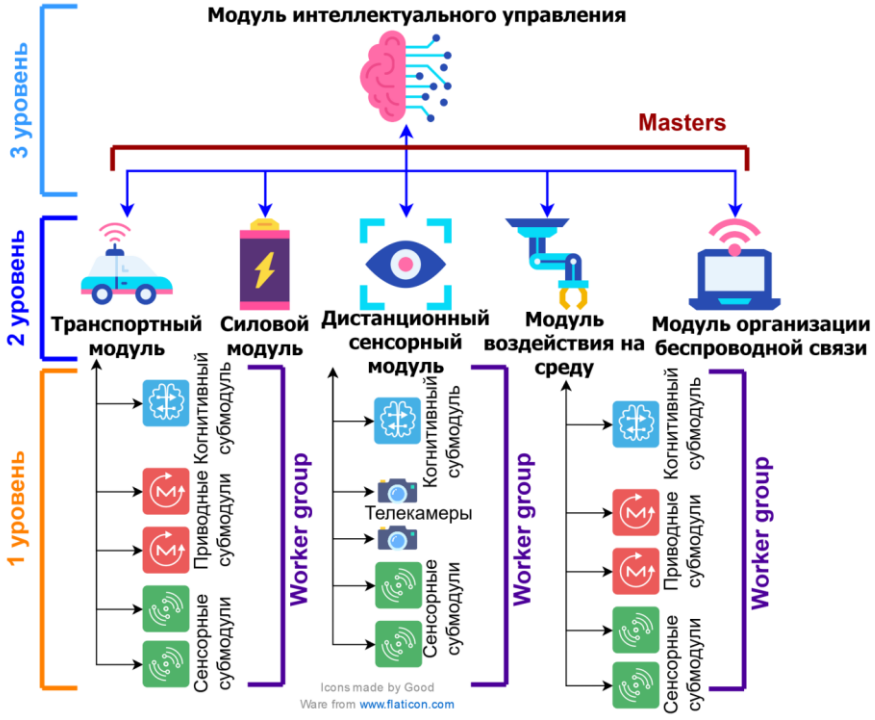

Рис. 1 Функционально-модульная архитектура системы управления мобильного робота

Fig. 1. Functional-modular architecture of a control system of a mobile robot

На вершине архитектуры располагается модуль интеллектуального управления, отвечающий за планирование и распределение задач по функциям между модулями второго

[1] ISO/DIS 22166-1:2021(E). 2021.

уровня – общесистемное управление. В этой архитектуре модули на третьем и втором уровне иерархии отвечают за одну укрупнённую функцию целого робота согласно принципу полной функциональности [5]: *каждый модуль робота должен быть способен любым удобным ему способом выполнять свою целевую функцию, используя только собственные средства для выполнения команд от внешней системы управления*. На первом уровне иерархии располагаются субмодули, выполняющие функцию низкоуровневого управления исполнительными устройствами, например, приводами; фактически это группа субмодулей (*Worker group*). Взаимодействие модулей второго и первого уровня выполняется по принципу «*Master-Worker group*».

Системы управления большинства современных МР строятся на основе одного вычислительного устройства, на котором выполняются все вычислительные операции (как высокоуровневого, так и низкоуровневого управления), что не допускает ***оперативную*** перенастройку СУ робототехнической системы; реализация перенастройки СУ таких роботов на практике означает создание нового робота с заданной новой функциональностью. Модульное решение СУ означает разбиение функционала РТК на более простые подфункции (рис.1), которые реализуются отдельными полнофункциональными мехатронными модулями, обладающими собственной системой управления [4]. Следует особо отметить, что для обеспечения минимизации массогабаритных параметров и энергопотребления автономных мобильных роботов системы управления модулей должны реализовываться на встраиваемых системах – микроконтроллерах невысокой производительности. Для синхронизации работы такой распределённой структуры необходимо обеспечить соответствующее межмодульное информационное взаимодействие, которое реализуется объединением систем управления всех модулей и субмодулей в локальную вычислительную сеть (ЛВС) [6]. В результате, реализуются распределённые вычисления по аналогии с мультиагентными системами [7].

В МР с модульной архитектурой целевая задача решается за счёт правильного подбора функциональных узлов-модулей и их автоматической программной интеграции в СУ робота в целом без переработки программных аспектов как общей СУ, так и СУ самих модулей. Однако, вследствие неопределённости требований к параметрам СУ в случаях, когда заранее невозможно определить условия окружающий среды, в которых предстоит работать роботу (например, разведывательный МР на иных планетах или в зонах техногенных катастроф), возникает необходимость перенастройки СУ или, даже, замены его программного обеспечения (ПО) – удалённой и управляемой извне адаптации, настройки и подстройки к ситуации. Возникает *необходимость оперативного дистанционного перепрограммирования* программных аспектов систем управления отдельных модулей робота без остановки функционирования РТК, поскольку в указанных условиях доставить робот его разработчику или изготовителю оказывается крайне затратным или даже невозможным. В то же время при наличии связи с роботом возможно дистанционно выполнить эту работу с помощью разработчика ПО.

Цель настоящей работы – анализ существующих и перспективных методов динамического перепрограммирования, пригодных для использования в мобильном роботе с модульной архитектурой системы управления. Основная проблема использования динамического перепрограммирования для модульного робота – необходимость использования в СУ модулей встраиваемых микроконтроллерных систем и обновление программной части «на лету» через основную шину данных, используемую микроконтроллером, без отключения и демонтажа блоков. Поскольку системы управления модулей и субмодулей объединены в ЛВС, то в основе решения данной задачи лежит поиск соответствующих методов межмодульной коммуникации (ММК) и сетевых протоколов, отвечающих определённым требованиям.

Авторами из разных стран и научных школ были предложены разные методы, как построения модульных роботов [8-10], так и методов межмодульной коммуникации [11-13]. В нашей работе [14] показано, что спецификация *Cyphal* [15] (ранее известная как *UAVCAN*) и её реализация *OpenCyphal* являются наиболее эффективными с точки зрения применения в качестве ММК. Поэтому, различные механизмы динамического перепрограммирования, рассматриваемые в данной статье, будут использовать в качестве ММК сообщения из спецификации *Cyphal*.

Материалы и методы

В рамках настоящей работы будем рассматривать встраиваемую (реализуемую на микроконтроллере) программно-аппаратную систему, которая управляет некоторым физическим процессом. Управление производится «основным алгоритмом». Любые изменения в основной алгоритм производятся «побочным алгоритмом».

На данный момент существует множество разных и значительно отличающихся друг от друга методов динамического перепрограммирования. Каждый из них имеет свои уникальные свойства и отличается разным распределением сложности задач по изменению «основного алгоритма» между человеком-разработчиком системы управления, человеком-настройщиком (конечным пользователем) системы управления, программным обеспечением на ЭВМ разработчика/настройщика встраиваемой системы и непосредственной системой управления встраиваемой системы. Отличия также заключаются в широте гибкости решений, которые возможно создать с помощью каждого из этих методов. Ниже рассмотрены 4 основных метода динамического перепрограммирования:

М1. Параметрическое перепрограммирование.

М2. Замена кода микроконтроллера.

М3. Замена байткода.

М4. Замена скрипта на интерпретируемом языке программирования.

Анализ каждого из перечисленных методов выполняется по следующим критериям:

К1. Применимость на разных встраиваемых системах – микроконтроллерах невысокой производительности.

К2. Объём требуемой оперативной памяти и памяти программ.

К3. Скорость вычислений (процессорное время).

К4. Теоретическая сложность создания реализации метода в виде базовой управляющей программы.

К5. Теоретическая сложность использования метода для адаптации к оперативной ситуации базовой управляющей программы конечным пользователем.

К6. Гибкость метода – глубина и сложность внесения возможных изменений в основной алгоритм работы встраиваемой системы при использовании данного метода.

Важным критерием для методов *дистанционного* динамического перепрограммирования является возможность обеспечения информационной безопасности, поскольку дистанционность обычно реализуется за счёт использования радиоканала. Рассмотренные ниже методы дают возможность удалённого исполнения кода (***Remote Code Execution***), что является одной из крупнейших уязвимостей для любого комплекса программ. Но авторы не являются специалистами в сфере информационной безопасности, поэтому обеспечение информационной безопасности представленных ниже методов – это тема для отдельного рассмотрения соответствующими специалистами. В данном случае считается, что все взаимодействия происходят внутри помехозащищённой внутренней шины робота, недоступной для внешних воздействий. Также ПО того узла внутренней сети, которое производит обмен информацией с внешними, потенциально открытыми сетями, написано достаточно надёжно, что не допускает загрузку и исполнение кода из недостоверных и потенциально вредоносных источников.

Результаты анализа

М1 – Параметрическое перепрограммирование

Одним из первых и самых простых методов динамического перепрограммирования является параметрическое перепрограммирование – то есть замена неизменных (с точки зрения основного алгоритма управления) внутренних величин, которые играют существенную роль в работе этого алгоритма. Пример – подстройка значений пропорциональной, интегральной и дифференциальной частей в регуляторе скорости вращения электродвигателя в системе управления модуля движения. Значения (например, по умолчанию) для таких параметров могут храниться в энергонезависимой памяти вычислительного устройства. Именно такой способ перепрограммирования предоставляют типы данных с доступом сервис (запрос-ответ): *uavcan.register.Access и uavcan.register.List* из спецификации *Cyphal.* Оба эти сервиса предоставляют информацию об именованных параметрах прошивки – так называемых регистрах. Каждый регистр обладает именем и значением. Список имён регистров можно получить с помощью последовательных вызовов службы *uavcan.register.List;* получить существующее или установить новое значение в именованный регистр можно с помощью команды *uavcan.register.Access*. Реализация *OpenCyphal* предоставляет возможность массовой установки данных параметров с помощью файла конфигурации, что облегчает настройку всей взаимодействующей системы целиком. Необходимо отметить, что использование регистров позволяет не только установить значения параметров отдельного алгоритма, но и предоставляет возможность «на лету» менять и сам основной алгоритм путём выбора одного из возможных сценариев.

Оценивая метод по критерию К1, можно отметить, что данный метод подходит для подавляющего большинства встраиваемых систем, так как он не подразумевает

модификацию энергонезависимой памяти вообще или модификацию лишь небольшой её части. Многие современные встраиваемые микроконтроллеры и встраиваемые системы или обладают возможностью записывать данные напрямую в энергонезависимую память программ, или содержат отдельную небольшую энергонезависимую встроенную область памяти, которая предназначена именно для хранения параметров. Если же микроконтроллер не обладает ни одной из перечисленных возможностей, то такая память может быть установлена извне.

С точки зрения потребления оперативной памяти и памяти программ (критерий К2), данный метод требует начального «вложения» в объёме реализаций метода приёма и передачи перечисленных ранее сообщений, метода доступа к энергонезависимой памяти, а также дополнительных затрат на хранение как названий параметров, так и самих параметров. Спецификация *Cyphal* предполагает максимальную длину сообщения на чтение и/или модификацию одного параметра в 515 байт. Из них 256 байт отведены на название регистра, а 259 байт – на его значение.

Дополнительная нагрузка также ложится на вычислительную сложность и самого основного алгоритма (критерий К3). Если бы параметры были установлены в тексте программы как константы, известные при создании машинных кодов, то открывается простор для оптимизаций, реализовать которые сложно или просто невозможно, если эти параметры могут меняться во время работы. Эту нагрузку сложно предсказать и нужно заранее замерять. На современных микроконтроллерах с широкой (32 бита) шиной данных, при использовании ограниченного количества параметров и хорошего оптимизирующего компилятора, можно предполагать ухудшение производительности не более чем на несколько процентов.

С точки зрения, как разработчика управляющей программы (критерий К4), так и пользователя (критерий К5), использование данного метода оказывается довольно простым. Разработчику достаточно добавить ещё одну функцию, обрабатывающую соответствующие сообщения. Пользователь же может использовать готовые интерфейсы, как командной строки, так и графические, чтобы настроить готовое устройство, не открывая его исходный код.

Основной недостаток этого метода в отсутствии гибкости. Невозможно адаптировать модуль к совершенно новому сценарию, который не был описан в исходном коде. Для работы данного метода нет необходимости останавливать работу микроконтроллера.

М2 – Замена кода микроконтроллера

Следующий метод динамического перепрограммирования основан на замене машинного кода, записанного в памяти программ во встраиваемой системе. Примеры реализации подобного подхода обсуждаются в [16, 17].

Для работы данного метода со спецификацией Cyphal нужно реализовать обработку сообщений для сервиса вызова обобщённых команд *uavcan.node.ExecuteCommand*.1.0. Одна из обобщённых команд – *COMMAND_BEGIN_SOFTWARE_UPDATE* (запуск обновления программ на встраиваемой системе). Имя файла с новой программой указывается в параметре запроса на эту команду. В ответ на эту команду узел сети совершает сначала запрос *uavcan.file.GetInfo* для получения информации о файле с новыми машинными кодами, а затем *uacan.file.Read* для получения этого файла по частям.

Существует несколько реализаций данного метода:

1. Отдельная программа – загрузчик.
2. Обновление внутри программы.

Загрузчик (М2.1)

Самым простым и одновременно сложным вариантом обновления машинного кода является использование отдельной программы – загрузчика. Эта программа вызывается средствами самой встраиваемой системы до загрузки основной программы и может выполнять базовую настройку микроконтроллера, а затем вызвать основную программу на исполнение. Вместе с этим у программы загрузчика есть более широкие возможности управления встраиваемой системой, в частности – возможность получить извне и перезаписать управляющую программу.

Далеко не все встраиваемые системы аппаратно поддерживают загрузку с использованием загрузчика (критерий К1). На таких устройствах работу загрузчика нужно либо восполнять программными средствами, либо не пользоваться загрузчиком вообще.

Использование загрузчика уменьшает максимальный объём памяти программ на размер самого загрузчика (критерий К2).

По критерию К3, использование загрузчика не влияет на работу основного алгоритма. Правильно написанный загрузчик полностью отдаёт управление основной подпрограмме, не оставляя следов в оперативной памяти или настройке всей периферии.

Основная сложность загрузчиков кроется, с точки зрения разработчика (критерий К4), в создании (а) достаточно надёжной и (б) небольшой программы. Первое – нужно, чтобы встраиваемая система оставалась в сети, несмотря на ошибки при передаче и/или записи в энергонезависимую часть памяти программ встраиваемой системы. Второе – нужно, чтобы загрузчик смог поместиться в ограниченную и небольшую область памяти программ.

С точки зрения пользователя (критерий К5) сложность этого метода довольно высока – нужно вносить правки напрямую в исходные тексты программ для встраиваемой системы, нужно знать язык программирования исходных текстов, нужно иметь среду программирования и достаточно производительный компьютер. Одна из основных проблем здесь – создание повторяемой у пользователя среды разработки со всеми необходимыми инструментами.

Данный метод предоставляет максимальную гибкость (критерий К6) в модификации и адаптации программного обеспечения, но зависит от поставки исходных кодов программ встраиваемой системы. Также большим как недостатком, так и достоинством данного метода является необходимость останавливать и перезагружать микроконтроллер для обновления программы. Основной алгоритм должен уметь адекватно реагировать на запросы на обновление программного обеспечения.

Обновление внутри программы (М2.2)

Метод обновления внутри программы и следующий метод – обновление части машинного кода – решают проблему обновления программного обеспечения путём сращивания основного и побочного алгоритмов. Обновление производится параллельно с работой основного алгоритма, машинные коды новой программы записываются в отдельную область памяти программ, она проверяется на ошибки и копируется вместо основной программы. После этого управляющая программа запускается вновь.

К сожалению, далеко не все встраиваемые системы могут использовать данный метод, так как для него необходимо иметь возможность менять память программ во время работы (критерий К1).

Данный метод несколько эффективнее использует память программ (критерий К2) за счёт использования тех же коммуникационных механизмов, что и основная программа. Однако основной недостаток данного метода заключается в максимальном объёме памяти программ, который программа может использовать – он сокращён в два раза, чтобы в любой момент работы алгоритм обновления мог записать в память программ очередной кусочек обновления.

Работа алгоритма обновления не занимает дополнительного процессорного времени (критерий К3) в режиме ожидания, а в режиме обновления может работать с низким приоритетом, не мешая режиму реального времени основной программы.

С точки зрения разработчика (критерий К4), реализация данного метода не несёт особой сложности – достаточно создать дополнительный обработчик команд и отдельный «поток», производящий обновление. Но, по сравнению с отдельным загрузчиком, сложность оказывается ниже за счёт снижения требований к объёму кода и наличия возможности разделить код сетевого взаимодействия с основной программой.

Для пользователя (критерий К5) использование данного метода по сложности сравнимо со сложностью метода с загрузчиком – также нужно получить правильно настроенную среду разработки, знать и уметь использовать язык программирования, на котором написаны исходные тексты. Однако, в связи с ограничением на размер прошивки, пользователь также может быть вынужден проводить дополнительную оптимизацию программы, чтобы она смогла поместиться в сокращённую вдвое область памяти программ.

Гибкость применения (критерий К6) данного метода выше, чем у метода с загрузчиком из-за меньшего времени простоя микроконтроллера – долгие операции получения новой прошивки интегрированы в основной алгоритм, основное время простоя происходит в момент копирования и перезапуска управляющей программы.

М3 – Замена байткода

Метод замены байткода основан на том, что на встраиваемой системе вместо основной программы запускается программный интерпретатор кодов, отличных от машинного кода самой встраиваемой системы, т.е. программный процессор байткода. На данный момент существует множество разных программных процессоров байткода со своими особенностями. Пример такого программного процессора приведён в работе [18].

Данный метод может использовать методы межмодульной коммуникации согласно спецификации *Cyphal* по аналогии с двумя предыдущими методами. В случае использования параметрического перепрограммирования, байткод управляющей программы записывается в параметр (регистр) типа «неструктурированный массив». Если же используется обновление через файл (см. прошлый раздел), то вместо обновления машинных кодов, производится обновление области памяти программ, отвечающей за хранение байткода.

С точки зрения производительности, программный процессор байткода будет всегда кратно медленнее, чем аналогичный код в машинных кодах. Сам программный процессор может занимать значительные объёмы памяти программ и данных. Небольшого улучшения скорости работы можно добиться за счёт переноса сложных вычислительных операций в машинный код программного процессора. Из-за этого данный метод может быть неприменим на встраиваемых системах с узкой (8 и 16 бит) шиной данных и небольшими (до 1 кБ) объёмами памяти программ и данных (критерии К1-К3).

С точки зрения разработчика (критерий К4) программного обеспечения, в данном методе основную сложность составляет выбор и перенос на встраиваемую систему существующего программного процессора, а также разработка дополнительных механизмов взаимодействия этого программного процессора с периферийными устройствами микроконтроллера. Также разработчик должен предоставить пользователю программу-транслятор какого-либо языка высокого уровня в байткод программного процессора. Если используется байткод существующего интерпретируемого языка программирования, то эта задача существенно упрощается.

Для пользователя (критерий К5) данный метод оказывается проще, чем использование замены машинных кодов (М2), но требует знания и использования какого-либо языка высокого уровня или даже программирования в байткодах. Однако цикл разработки существенно упрощается, так как программа-транслятор с языков высокого уровня может быть гораздо проще в настройке и установке, чем наборы инструментов для работы с машинным кодом встраиваемой системы.

Гибкость (критерий К6) данного метода существенно ниже, чем у метода замены машинных кодов из-за меньшего объёма возможных алгоритмов: сложный алгоритм в байткоде может не уложиться в ограничение на скорость работы в реальном времени, а тот же алгоритм, преобразованный в машинный код и интегрированный в программный процессор, может не дать ту гибкость подстройки, которую дала бы полная замена машинных кодов. Из-за этого разработчику приходится балансировать в месте реализации алгоритмов основной программы (байткод или машинный код), а пользователь вынужден мириться с этим балансом при адаптации своих алгоритмов.

М4 - Замена скрипта на интерпретируемом языке программирования

Данный метод базируется на методе замены байткода и имеет схожие характеристики. В этом методе на встраиваемую систему присылается не байткод, а исходный код на языке высокого уровня, который далее исполняется напрямую или с использованием трансляции в байткод. Пример такого интерпретатора приведён в работе [18].

Данный метод требует более высокой производительности встраиваемой системы – необходимо выполнить такой же или больший объём вычислений, по сравнению с программным процессором байткода (критерии К1, К3). Из-за необходимости разбирать текст языка программирования для прямой интерпретации или преобразования в байткод, повышаются объём машинного кода и объём данных. Также из-за использования текста в качестве программы повышаются требования к объёму памяти интерпретируемых программ (критерий К2).

По сравнению с байткодом на плечи разработчика (критерий К4) также ложится перенос и оптимизация программы-транслятора на встраиваемой системе. Также в данном методе повышаются требования к механизмам обработки и информирования об ошибках – пользователь должен получать сообщения, если в его программе допущены какие-либо синтаксические или логические ошибки.

С точки зрения пользователя (критерий К5) данный метод несколько сложнее параметрического перепрограммирования, т.к. подразумевает владение языком программирования высокого уровня. Однако же этот метод существенно проще метода замены байткода – нет необходимости устанавливать программу-транслятор.

В смысле гибкости (критерий К6) данный метод сравним с методом замены байткода. Однако, из-за необходимости работать с текстовым представлением команд

разработчику приходится ещё усерднее искать баланс между алгоритмами, закладываемыми в машинный код, и алгоритмами, предоставляемыми пользователем.

Заключение

На данный момент существует множество разных способов динамического перепрограммирования. Среди представленных в статье, наиболее простым в применении является метод М1 – параметрическое перепрограммирование. Метод реализуем на любых встраиваемых системах, в отличие от остальных методов, которые требуют или аппаратной поддержки некоторых функций (замена машинного кода – М2), и/или большого объёма памяти программ и данных (замена байткода (М3), или программы на интерпретируемом языке программирования (М4)), однако он наименее гибок. Самыми простыми для пользователя являются методы М1 и М4, так как не требуют специального программного обеспечения или дополнительных знаний. Метод М2 идеален для случаев использования встраиваемых систем с наименьшим объёмом памяти программ и данных. Метод М3 хорош в том случае, если необходима гибкость метода М4, но применить М4 не позволяет использование встраиваемой системы, которая не обладает соответствующими параметрами.

Кроме проведённого анализа перечисленных методов по шести критериям предполагается в перспективе экспериментально проверить реализуемость каждого метода на микроконтроллерах семейств *STM32F*103, *GD*32*VF*103, *ESP*32 и микрокомпьютерах *Raspberry Pi u Orange Pi*.

Литература

1. Попов Е.П., Письменный Г.В. Основы робототехники: введение в специальность. – М.: Высшая школа, 1990. 224 с.

2. Лопота А.В., Юревич Е.И. Этапы и перспективы развития модульного принципа построения робототехнических систем // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2013. № 1. С. 98-103.

3. Платонов А.К. Робототехника лунной базы // XXXIV Чтения по космонавтике. ИПМ им. М.В. Келдыша РАН, 2010.

4. Андреев В.П., Ким В.Л., Эприков С.Р. Аппаратно-программный фреймворк для разработки модульных мобильных роботов с иерархической архитектурой // Известия ЮФУ. Технические науки. Раздел IV. Связь, навигация и наведение. Таганрог: Изд-во ФГАОУ ВО Южный федеральный университет, ISSN 1999-9429. 2020. №1(211). С. 199-218.

5. Андреев В.П., Ким В.Л., Плетенев П.Ф. Принцип полной функциональности модулей в гетерогенных модульных мобильных роботах / Экстремальная робототехника (ЭР-2017). Труды международной научно-технической конференции. Санкт-Петербург: ИПЦ ООО «Политехника-принт», 2017. С.81–91.

6. Андреев В.П., Плетенев П.Ф. Метод информационного взаимодействия для систем распределённого управления в роботах с модульной архитектурой // Труды СПИИРАН. 2018. № 2 (57). С. 134-160.

7. Андреев В.П. Система управления модульных мобильных роботов как мультиагентная система с пирамидальной топологией // "Известия высших учебных заведений. Северо-Кавказский регион. Технические науки", ISSN 1560-3644. 2020. № 3(207). С. 41-54.

8. S. Herbrechtsmeier, T. Korthals, T. Schopping, U. Ruckert AMiRo: a modular & customizable open-source mini robot platform // 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia. 2016. pp.687-692.

9. Design of Transmote: a Modular Self-Reconfigurable Robot with Versatile Transformation Capabilities / Guifang Qiao, Guangming Song, Jun Zhang, Hongtao Sun, Weiguo Wang & Aiguo Song // Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics. 2012. pp.1331-1336.

10. Baca J., Ferre M., Aracil R. A heterogeneous modular robotic design for fast response to a diversity of tasks // Robotics and Autonomous Systems. 2012. vol. 60. no. 4. pp. 522–531.

11. R2P: An open source hardware and software modular approach to robot prototyping / A. Bonarini, M. Matteucci, M. Migliavacca, D. Rizzi // Robotics and Autonomous Systems. 2014. No.62. pp.1073-1084.

12. Distributed and modular CAN-based architecture for hardware control and sensor data integration / D.P. Losada, J.L. Fernández, E. Paz, Rafael Sanz // Sensors. 2017. No.17. pp.1013-1030.

13. L. Peng, F. Guan, L. Perneel, H. Fayyad-Kazan and M. Timmerman EmSBoT: A lightweight modular software framework for networked robotic systems // 2016 3rd International

Conference on Advances in Computational Tools for Engineering Applications (ACTEA), Beirut, 2016, pp.216-221. doi: 10.1109/ACTEA.2016.7560142.

14. Andreev, Victor & Pletenev, Pavel. Problems of Choosing an Intermodule Information Interaction Protocol for Mobile Robots with Modular Control System Architecture, Proceedings of the 32nd DAAAM International Symposium, pp.0151-0157, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-33-4, ISSN 1726-9679, Vienna, Austria.

15. Kirienko P., Dixon S, et al. OpenCyphal: Open technology for real-time intravehicular distributed computing and communication based on modern networking standards. URL: https://opencyphal.org/specification/Cyphal_Specification.pdf.

16. Lobdell M. Robust over-the-air firmware updates using program flash memory swap on kinetis microcontrollers // Freescale Application Note, p. AN4533. 2012.

17. Jaouhari S. E., Bouvet E. Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions // Internet of Things. 2022. (18). C. 100508.

18. Zandberg K., Baccelli E. Minimal virtual machines on IoT microcontrollers: The case of berkeley packet filters with rbpf // arXiv preprint arXiv:2011.12047. 2020.

Авторы публикации

Андреев Виктор Павлович – д-р техн. наук, профессор кафедры «Сенсорные и управляющие системы» Московского государственного технологического университета «СТАНКИН» (МГТУ «СТАНКИН»).

Плетенев Павел Филиппович – аспирант Московского государственного технологического университета «СТАНКИН» (МГТУ «СТАНКИН»), инженер АО «ВНИИЖТ».

References

1. Popov E.P., Pis'mennyi G.V. *Osnovy robototekhniki: vvedenie v spetsial'nost'*. M.: Vysshaya shkola, 1990. 224 p.

2. Lopota A.V., Yurevich E.I. Etapy i perspektivy razvitiya modul'nogo printsipa postroeniya robototekhnicheskikh system. *Nauchno-tekhnicheskie vedomosti SPbGPU. Informatika. Telekommunikatsii. Upravlenie*. 2013. № 1. Pp. 98-103.

3. Platonov A.K. Robototekhnika lunnoi bazy. XXXIV Chteniya po kosmonavtike. IPM im. M.V. Keldysha RAN, 2010.

4. Andreev V.P., Kim V.L., Eprikov S.R. Apparatno-programmnyi freimvork dlya razrabotki modul'nykh mobil'nykh robotov s ierarkhicheskoi arkhitekturoi. *Izvestiya YuFU. Tekhnicheskie nauki. Razdel IV. Svyaz', navigatsiya i navedenie.* Taganrog: Izd-vo FGAOU VO Yuzhnyi federal'nyi universitet, ISSN 1999-9429. 2020. №1(211). p. 199-218.

5. Andreev V.P., Kim V.L., Pletenev P.F. *Printsip polnoi funktsional'nosti modulei v geterogennykh modul'nykh mobil'nykh robotakh.* Ekstremal'naya robototekhnika (ER-2017). Trudy mezhdunarodnoi nauchno-tekhnicheskoi konferentsii. Sankt-Peterburg: IPTs OOO «Politekhnika-print», 2017. pp.81-91.

6. Andreev V.P., Pletenev P.F. *Metod informatsionnogo vzaimodeistviya dlya sistem raspredelennogo upravleniya v robotakh s modul'noi arkhitekturoi. Trudy SPIIRAN*. 2018. № 2 (57).pp. 134-160.

7. Andreev V.P. Sistema upravleniya modul'nykh mobil'nykh robotov kak mul'tiagentnaya sistema s piramidal'noi topologiei. *Izvestiya vysshikh uchebnykh zavedenii. Severo-Kavkazskii region. Tekhnicheskie nauki"*, ISSN 1560-3644. 2020. № 3(207). pp. 41-54.

8. Herbrechtsmeier, T. Korthals, T. Schopping, U. *Ruckert AMiRo: a modular & customizable open-source mini robot platform.* 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia. 2016. pp.687-692.

9. *Design of Transmote: a Modular Self-Reconfigurable Robot with Versatile Transformation Capabilities.* Guifang Qiao, Guangming Song, Jun Zhang, Hongtao Sun, Weiguo Wang & Aiguo Song. Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics. 2012. pp.1331-1336.

10. Baca J., Ferre M., Aracil R. A heterogeneous modular robotic design for fast response to a diversity of tasks. *Robotics and Autonomous Systems*. 2012. vol. 60. no. 4. pp. 522–531.

11. R2P: An open source hardware and software modular approach to robot prototyping / A. Bonarini, M. Matteucci, M. Migliavacca, D. Rizzi . *Robotics and Autonomous Systems.* 2014. No.62. pp.1073-1084.

12. D. P. Losada, J. L. Fernández, E. Paz, Rafael Sanz Distributed and modular CAN-based architecture for hardware control and sensor data integration. *Sensor*s. 2017. No.17. pp.1013-1030.

13. EmSBoT: A lightweight modular software framework for networked robotic systems / L. Peng, F. Guan, L. Perneel, H. Fayyad-Kazan and M. Timmerma. 2016 3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA), Beirut, 2016, pp.216-221. doi: 10.1109/ACTEA.2016.7560142.

14. AndreevVictor & Pletenev Pavel. Problems of Choosing an Intermodule Information Interaction Protocol for Mobile Robots with Modular Control System Architecture, Proceedings of the 32nd DAAAM International Symposium, pp.0151-0157, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-33-4, ISSN 1726-9679, Vienna, Austria.

15. Kirienko P, Dixon S, et al. OpenCyphal: Open technology for real-time intravehicular distributed computing and communication based on modern networking standards. URL: https://opencyphal.org/specification/Cyphal_Specification.pdf.

16. Lobdell M. Robust over-the-air firmware updates using program flash memory swap on kinetis microcontrollers // Freescale Application Note, p. AN4533. 2012.

17. Jaouhari S. E., Bouvet E. Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions // Internet of Things. 2022. (18). C. 100508.

18. Zandberg K., Baccelli E. Minimal virtual machines on IoT microcontrollers: The case of berkeley packet filters with rbpf // arXiv preprint arXiv:2011.12047. 2020.

Authors of the publication

Victor P. Andreev – Moscow State University of Technology "STANKIN", Moscow, Russia.

Pavel F. Pletenev – Moscow State University of Technology "STANKIN", Moscow, Russia, engineer at AO “VNIIZhT”, Moscow, Russia.

Получено ***23.05.2022г.***

Отредактировано ***30.05.2022г.***

Принято ***30.05.2022г.***