Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Кирсанов Александр Юрьевич

Программирование роботов с помощью .NET/Mono

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А.Н.

Научный руководитель:
ст. пред. Кириленко Я.А.

Рецензент:
к.ф.-м. н. МАИ, Майкрософт Сошников Д.В.

Санкт-Петербург
2015

SAINT PETERSBURG STATE UNIVERSITY

Chair of Software Engineering

Aleksandr Kirsanov

Robots Programming using .NET/Mono

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terehov

Scientific supervisor:
senior lecturer Iakov Kirilenko

Reviewer:
assosiate professor MAI, Microsoft Dmitri Soshnikov

Saint Petersburg
2015

Оглавление

Введение

Развитие контроллеров

Технические характеристики современных контроллеров позволяют заметно расширить как области применения таких устройств, так и набор доступных для их программирования языков и технологий. С увеличением мощностей, времени автономной работы и уменьшением размеров роботам находят новые применения в не только в промышленности, но и в самых разных областях.

Отдельно стоит отметить рост персональной робототехники и её заслуги в STEM[9] обучении. На протяжении многих лет роботизированные контроллеры служат отличным способом привить детям интерес к математике, программированию и естественным наукам. Существует множество проектов, созданных специально для обучения: LEGO Mindstorm, Gumstix, TRIK.

Появляется всё больше контроллеров для персонального использования. Доступность таких систем привлекает энтузиастов, радиолюбителей и рядовых программистов, не знакомых с разработкой встраиваемых систем. Повышение уровня абстракции делает разработку доступной не только для узкого класса специалистов.

Мы наблюдаем качественное изменение технологий, создаваемых для программирования роботов. Увеличение размеров программ, а также количества людей, вовлечённых в проекты, связанные с программированием роботов, приближаются к показателям остальных областей разработки ПО. Размеры и сложность программ, создаваемых для роботов, требуют специальной организации и использования инструментов, аналогичных тем, которые используются в промышленной разработке.

Опыт индустрии промышленного программирования

За время своего существования индустрия производства ПО выработала широкий набор инструментальных средств, к которым можно отнести системы тестирования, статические анализаторы, инструменты профилирования и отладки, профессиональные текстовые редакторы и IDE. Для каждого языка или фреймворка такие инструменты формируют определённую рабочую инфраструктуру, которой будут пользоваться программисты. Её целостность и уровень развития позволяют ускорить процесс разработки всего продукта и оказывают серьёзное влияние на популярность самой платформы. Необходимость в использовании различных инструментов увеличивается вместе с размерами проектов, поэтому поддержка со стороны инструментов становится актуальной проблемой в разработке ПО для роботов.

Повторное использование ПО

Кроме создания новых технологий для разработки роботов становится возможным повторно использовать существующие языки, библиотеки и целые платформы. Успешный опыт такого подхода можно наблюдать в мобильной индустрии. Java используется в качестве основного языка для разработки пользовательских приложений для устройств под управлением Android[1]. Платформа Windows Phone использует адаптированную версию .NET, а с помощью Xamarin[2] (.NET/Mono[3]) можно создавать кросс-платформенные мобильные приложения.

Повторное использование позволяет привлечь в новую область людей, уже знакомых с технологией, таким образом обеспечив новую индустрию рабочими кадрами. Кроме того, наличие хорошей профессиональной литературы позволяет упростить процесс обучения новых специалистов. Но пожалуй самым важным преимуществом данного подхода является то, что при удачном выборе платформы становится возможно переиспользовать созданные для неё инструментальные средства и библиотеки.

Всё эти аспекты очень актуальны для робототехники, как для набирающей популярность области программирования. Такой подход позволит:

- расширить набор языковых средств используемых для роботов, к которым ещё недавно относились только язык ассемблера C и C++, высокоуровневым программированием и современными технологиями;
- ещё больше приблизить программирование роботов к остальным областям разработки ПО;
- восполнить недостаток библиотек и хороших инструментальных средств, которые необходимы при текущих размерах проектов;
- снизить порог вхождения для такой разработки, что особенно важно при росте популярности персональной робототехники.

[1] www.android.com
[2] www.xamarin.com
[3] www.mono-project.org

1. Постановка задачи

Принимая во внимание успешные проекты, повторно использующие технологии в новых областях, а также то, что многие компиляторы и среды исполнения были успешно портированы на ARM[4], повторное использование технологий в робототехнике можно считать обоснованным.

Основной задачей данной работы является адаптация .NET для разработки ПО роботов. Взяв за основу существующую и хорошо развитую систему библиотек и инструментальных средств платформы .NET, необходимо дополнить её библиотеками и инструментами, учитывающими особенности разработки роботов и встраиваемых систем.

[4]www.arm.com

2. Существующие решения

Существует несколько проектов, позволяющих использовать .NET для программирования встраиваемых устройств и роботов. Сами проекты очень отличаются друг от друга и созданы для совершено разных задач. Попытки сопоставить существующие решения с требованиями либо выявляют между ними крупные несоответствия, либо вскрывают недостатки существующих реализаций. В данном разделе приведён краткий обзор технологий, а так же причины, по которым конкретные проекты не подходят для решения поставленной задачи.

2.1. Microsoft Robotics Studio

Первым проектом для рассмотрения стала интегрированная среда разработки роботов Robotics Studio[5], созданная компанией Microsoft. Для среды существует визуальный язык, она предоставляет возможность вести разработку на языках семейства .NET, а также средства 2D эмуляции будущих алгоритмов.

Несмотря на кажущиеся преимущества, это решение сложно назвать подходящим для нашей задачи. В качестве контроллера или исполнителя для проектов данной платформы чаще всего выступает ноутбук, подключённый к Kinect[6] сенсору. Существуют специальные подвижные столики с электромоторами, на которые будет крепиться такой компьютер. Помимо этого платформа предоставляет средства интеграции с устройствами под управлением Windows CE, опять же с обработкой данных на ПК. В связи с ростом популярности робототехники и развитием встраиваемой техники готовым решением можно было бы считать среду для запуска на автономных современных контроллерах, поэтому данный проект нам не подходит.

2.2. Netduino и Microframework

Для втраиваемых устройств существует специальная версия .NET фреймворка, которая называется Micro Framework. Среди контроллеров, использующих данную технологию, самым известным проектом является netduino[7].

Microframework[8] — минималистичная версия .NET, созданная специально для работы на маловычислительных контроллерах. Образ рантайма со всеми включёнными во фреймворк библиотеками занимает на плате всего 250кБ. Сам фреймворк не требует для запуска работающей ОС, при этом частично выполняя её функции. MF предоставляет средства для обработки прерываний и доступ к UART, I^2C и другим

[5]https://msdn.microsoft.com/en-us/library/bb648760.aspx
[6]https://www.microsoft.com/en-us/kinectforwindows/
[7]www.netduino.com/
[8]https://netmf.codeplex.com

шинам. В то же время он может быть запущен с ОС, более того, для некоторых задач наличие ОС на роботе необходимо. MF использует некоторые примитивы ОС, в частности диспетчеризацию потоков.

Использование MF серьёзно сужает доступные программные средства средства:

- Программирование для такой платформы может везтись только на C#;
- В среде отсутствуют generics[12], являющиеся большим преимуществом .NET.;
- В состав MF входят только небольшая часть .NET фреймворка;
- Для сокращения затрат на JIT-компиляцию единственным возможным видом исполнения выбрана интерпретация;

Такой способ выполнения, хоть и снижает нагрузку на систему, накладывает серьёзные ограничения на скорость работы пользовательских программ. Стоит отметить и чрезвычайно скромные технические характеристики контроллеров, которые находятся под управлением MF. Использование таких контроллеров в качестве основы для разработки лишает смысла создание и адаптацию инстументальных средств. Контроллеры и программная платформа просто не смогут решать задачи, для которых эти инструменты будут полезны.

Сложно назвать эти факторы недостатками MF. Фреймворк разрабатывался и предназначен для запуска на устройствах с как минимум 64кБ оперативной памяти[9]. И нужно отметить успешно с этим справляется. Но в контексте текущей задачи эти факторы лишают смысла использование MF. Задачи, которые решают современные роботы, требуют более мощных контроллеров и полнофункциональной среды выполнения.

2.3. Raspberry Pi, TRIK

К другому классу устройств можно отнести контроллеры Rapberry Pi[10] и TRIK[11], которые оснащены мощными ARM процессорами, на которых работают специализированные дистрибутивы ОС Linux. Кроме ARM ТРИК оснащён ещё двумя процессорами, которые используются для вычислений и управления моторами, а Raspberry Pi часто используется совместно с другими контроллерами, что позволяет ещё больше увеличить вычислительные мощности устройства. Выбор Linux в качесте ОС позволяет использовать привычные разработчикам инструменты и языки программирования. Множество компиляторов, интерпретаторов и других программных средств, напрямую не связанных с языками программирования, либо изначально разрабатывались

[9] http://www.netduino.com/hardware/
[10] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
[11] http://www.trikset.com

кросс-платформенными, либо были портированны под ARM. В том числе и Mono — открытая реализация .NET. В состав Mono входит среда выполнения, компилятор C#, компилятор и интерпретатор F#. Среда выполнения arm-mono не отличается от среды выполнения x86 ничем кроме этапа кодогенерации в JIT и в отличие от MF никак не ограничивает выбор доступных библиотек и средств самой платформы.

Характеристики Raspberry Pi и TRIK позволяют использовать .NET в качестве программной платформы. Однако для .NET не существует библиотек и инструментов обеспечивающих работу с этими устройствами. Следует отметить, что данная проблема не является специфичной для самого .NET или представленных контроллеров. Общая практика использования современных языков на контроллеров, только набирает популярность, поэтому сложно выделить существование хоть сколько-нибудь общепринятой библиотеки для разработки роботов с помощью современной языковой платформы.

2.4. Выводы

Таким образом, не одно из существующий решений не отвечает поставленным требованиям и в полной мере не реализует возможности, доступные для разработки роботов на .NET. Поэтому создания такого решения является актуальной задачей.

3. Выбор аппаратной платформы

Следующие разделы настоящей работы будут посвящены различным этапам разработки, первым из которых является выбор подходящей аппаратной платформы. Идеи, стоящие в основе описываемой работы, не привязаны к конкретным платам и производителям, хотя и накладывают определённые ограничения на технические характеристики самих устройств, поэтому в ходе проекта хотелось бы поддержать работу с несколькими контроллерами.

Использование высокоуровневых языков и стремление приблизить разработку роботов к другим областям производства ПО подразумевает, что встраиваемые системы могут предложить мощности процессоров и объёмы памяти, необходимые для запуска и эффективного использования всех возможностей программной инфраструктуры. Эти ограничения позволяют выделить несколько контроллеров, подходящих для разработки.

В разделе, посвящённому обзору существующих решений, мы рассмотрели Raspberry Pi и TRIK. Эти контроллеры не предлагают стандартных библиотек и инструментов для программирования на .NET и поэтому полностью не решают поставленную задачу, но позволяют использовать .NET для разработки и поэтому интересны в качестве аппаратной платформы. Контроллеры послужат основой для инфраструктуры разработки роботов, которую мы дополним необходимыми библиотеками и инструментальными средствами.

4. Поддержка со стороны библиотек

4.1. Особенности задач, решаемых роботами

Современные роботы используются в самых разных областях. А решение задач, которые встают перед такими устройствами, включает использование сложных архитектурных и технологических приёмов. Контроллеры работают совместно с облачными вычислителями, которым делегируется выполнение ресурсоёмких задач по анализу данных и управлению[4]. Автономные роботы используют сложные математические вычисления для работы с видео, навигации и балансировки. Во многих коллективных соревнованиях таких как робофутбол, а так же при решении таких практических задач как мониторинг территорий используются целые команды роботов, которые обмениваются данными и выстраивают общую стратегию поведения

4.2. Реализация необходимых библиотек

Создание платформы для программирования роботов подразумевает анализ всех возможных сценариев использования, выделение типовых методов и создание библиотек для математики, работы с сетью, интеграцией с облачными сервисами и т.д. Однако при выборе .NET в качестве такой платформы мы можем использовать существующие библиотеки .NET фреймворка. С самом фреймворке содержится большое количество модулей, предназначенных для решения широкого круга задач. Кроме встроенных библиотек пользователь может использовать сторонние библиотеки, большинство из которых могут быть с лёгкостью установлены с помощью пакетного менеджера.

Количество реализованных для .NET фреймворка встроенных и сторонних библиотек позволяет решать даже специфичные для робота задачи, связанные с математическими расчётами и коммуникацией. Недостающей компонентой остаётся библиотека для доступа к периферийным устройства и управления самим контроллером. Реализация этой компоненты станет главной частью данного этапа проекта.

Первым контроллером для исследования стал ТРИК, выбор которого обусловлен удобством и доступностью таких контроллеров из-за сотрудничества проекта ТРИК с математико-механическим ф-м СПбГУ.

5. Библиотека Trik-Sharp

5.1. Контроллеры TRIK

Контроллеры ТРИК обладают развитой системой портов и слотов-расширений, которые позволяют подключать множество различных устройств среди которых: силовые моторы и сервоприводы; различные аналоговые и цифровые датчики (инфракрасные, ультрозвуковые, датчики расстояний, касания и т.д.); устройства, подключаемые через USB (например видеокамера). Кроме возможности подключения внешних сенсоров ТРИК предлагает набор встроенных компонентов. На корпусе самого контроллера можно найти кнопки, динамик и светодиодную лампу, а внутри платы расположены трехосевые гироскоп и акселерометр.

5.2. Основные возможности библиотеки

Основная задача библиотеки предоставлять доступ к устройствам на контроллере. В настоящий момент библиотека поддерживает работу со всеми вышеперечисленными внутренними и внешними периферийными устройствами, а также предоставляет средства для расширения существующего набора силами пользователя.

Помимо различных физических датчиков в библиотеке поддержана работа и с детекторами объектов и линии, работа которых осуществляется на DSP[12] путём анализа изображений, полученных с камер. Предоставляемый интерфейс помогает использовать технологии компьютерного зрения единообразно со всеми остальными устройствами.

Кроме работы с устройствами и видео библиотека предоставляет возможности для работы с самим контроллером. Существует модуль для работы с Shell, который может быть использован для запуска приложений и настройки робота. Модуль для работы с I^2C позволяет отправлять и получать данные через шину межпроцессорного взаимодействия. Через модуль мультимедиа пользователь получает возможность делать фотографии с камер, снимки с экрана на самом роботе, а также проигрывать файлы и пользоваться синтезом речи.

5.3. Архитектура

Библиотека содержит несколько пространств имён, в которых находятся классы и структуры, используемые в качестве данных от сенсоров; средства для работы с сетью, доступа к Shell и I^2C и т.д.

Основным классом для работы с контроллером является Model, который предоставляет наиболее общую конфигурацию будущего робота. Через свойства и методы

[12]http://www.ti.com/lit/ds/symlink/omap-l138.pdf

этого класса пользователи могут получить доступ к всем представленным устройствам.

Как правило, конкретные модели роботов не используют все доступные устройства. Если речь идёт о подключаемых устройствах, например о моторах или а камерах, то это означает, что эти устройства не будут подключены к контроллеру, а попытка инициализации приведёт к ошибке. Поэтому Model использует небольшие хитрости, для того, чтобы классы не создавались без необходимости, а на самом деле даже не JIT-компилировались, а именно тип lazy.

Пример

var model = new Model(); // Инстанцирование класса

Console.WriteLine(model.Accel.Read()); // Вывод значения с акселерометра

model.Led.SetColor(LedColor.Green); // Изменение цвета лампочки на зелёный

5.4. Детали реализации

Библиотека Trik-Sharp написана на F#, за исключением небольшого модуля для работы с I^2C, который реализован на C, а из библиотеки доступен через P/Invoke [6][10] и небольшие обёртки, скрывающие вызов unmanaged кода.

Вместе с исходным кодом пользователи могут познакомится с набором примеров, написанных на разных языках (C#, F#) и раскрывающих базовую функциональность библиотеки, методы работы с устройствами и архитектуру. Кроме небольших базовых примеров в репозитории содержится код различных моделей, настоящих роботов, собранных с помощью конструктора ТРИК. Модели, полностью написанные на .NET с помощью библиотеки Trik-Sharp, многократно использовались на различных выставках.

Языки платформы .NET используют одно и то же промежуточное представление MSIL[5]. Этот байт-код выполнятся CLR (Common Language Runtime), а компиляторы языков семейства .NET транслируют исходный код в это промежуточное представление. Работа со сборками (assembly), классами, исключениями и подавляющим большинством остальных особенностей .NET осуществляется именно на уровне байт-кода, поэтому классы из одних языков можно использовать в других, все эти черты формируют Common Type System (CTS) Тем не менее, обилие языков, каждый из которых обладает своей спецификой, вызывают определённые сложности при поддержке их совместимости. При разработке Trik-Sharp особое внимание было уделено использованию библиотеки из разных языков программирования. В следующем разделе мы рассмотрим некоторые из решений, сделанных в пользу улучшения этих показателей.

5.5. Некоторые сложности

Трудности вызваны стремлением сделать библиотеку удобной в использовании из C#, но при этом оставить её привычной для F# разработчиков. Описываемые решения связаны с особенностями F# и отличия средств используемых в разных языках.

Например с правилами наименования, которые в разных языках различаются. В F#, как и других языках семейства ML и в частности OCaml[13] основными единицами организации кода являются модули, а функции внутри модулей именуются с маленькой буквы. В случае с F# модули транслируются в статические классы, а функции из модуля — в статические методы, которые принято называть с большой буквы. В библиотеке учтены эти особенности с помощью системы атрибутов.

Shell.post "echo 0" // Отправка команды в F#

Helpers.Shell.Post("echo 0"); // Отправка команды в C#

Но гораздо более серьёзной проблемой является то, что стандартные для F# типы зачастую никак не соотносятся с классами из .NET. Хоть соответствующие классы созданы для аналогичных задач. Вызвано это прежде всего функциональной природой F#. В случае с коллекциями в F# сделан выбор в сторону неизменяемых аналогов, часто проигрывающим по эффективности стандартным средствам, но идеологически более правильными. Вот небольшой список стандартных для F#, но не для остального мира .NET практик:

- Активное использование кортежей, которые входят в состав .NET, но не часто используются.
- Использование каррированных функций (в некоторых версиях рантайма к тому же представленных отдельным классом FSharpFunc вместо System.Func)
- Использование функциональных списков, ref-объектов.

При разработке библиотеки очень важно соблюдать баланс в используемых средствах для того, чтобы сделать её использование максимально удобным из разных языков. Для F# существует набор правил[3] и практик для написания vanila-библиотек (направленных для использование из других языков). наравне с аналогичными практиками[1] для других .NET языков и способов поддержки работы между ними.

Public методы и свойства не содержат F# специфичных типов, поэтому при разработке пользователям даже не нужно подключать соответствующие F# сборки, которые понадобятся только при запуске.

[13] http://caml.inria.fr/ocaml/

5.6. Результаты

Пользователь не ограничивается в выборе библиотек и помимо представленной функциональности может использовать средства, предлагаемые .NET фреймворком и созданными для него сторонними библиотеками. Тот факт, что .NET использует платформо-независимый байт-код и обилие существующих библиотек поможет заметно сократить процесс разработки.

6. Оптимизации и эвристики

При использовании такой большой среды как .NET, особенно в условиях жёстких ограничений, стоит позаботиться о скорости работы библиотеки и среды выполнения. Несмотря на то, что контроллеры позволяют использовать такие высокоуровневые технологии, лишняя трата ресурсов непозволительна. Несмотря на то, что этот раздел не входил в круг задач, изначально поставленный перед автором, без проделанных оптимизаций было бы сложно представить завершения проекта.Технические характеристики заставляют гораздо острее чувствовать проблемы, с которыми борются и других .NET приложениях. В этом разделе мы рассмотрим проблемы связанные с производительностью, а также методы их решения.

6.1. Memory traffic

Прежде всего эти проблемы связаны с memory-traffic'ом, произвольным boxing'ом и, как следствие, лишней нагрузкой на сборщик мусора. Несмотря на то, что научные исследования и технологии в области сборки мусора сильно продвинулись и можно можно воспользоваться сборщиками с поколениями и отдельными кучами для больших объектов (LOH, Large Object Heap), серверными сборщиками, многопоточными сборщиками и т.д. Одно остаётся неизменным, сборка мусора — это трата ресурсов компьютера. Проблемы с большим количеством объектов и нежелательными сборками мусора испытывают многие приложения, в случае с роботами эта проблема кажется ещё более серьёзной. Во-первых контроллеры медленнее, чем современные ПК и серверные компьютеры, на которых запущены .NET приложения, поэтому сборка мусора происходит дольше. Во-вторых, задержки для таких устройств могут оказаться фатальными, потому что будучи системами, физически взаимодействующими с внешним миром, роботы могут причинить вред своей конструкции или, что ещё хуже, окружающим людям. В-третьих, роботы генерируют очень много объектов, например большая часть сенсоров шлёт данные с частотой 50Hz, а для многих устройств эту частоту можно увеличить. Таким образом, сборка мусора является очень важной проблемой при разработке библиотеки и пользовательских приложений в управляемой среде.

Во время разработки почти все классы, представляющие данные с сенсоров, были переписаны на структуры, которые не нагружают аллокатор и сборщик мусора. Часть логики библиотеки является реактивной, в частности все сенсоры. Реактивное программирование[7] накладывает несколько другие ограничения в плане сборки мусора[11]. В библиотеке используются результаты исследования и стандартная библиотека F# для работы с Observable-последовательностями.

6.2. JIT-компиляция

Пожалуй ещё большей проблемой стала так хорошо зарекомендовавшая себя в остальных областях промышленного программирования JIT-компиляция. Небольшие вычислительные мощности контроллера, а также большое количество сторонних библиотек, использовавшихся в первых версиях библиотеки, приводили к очень медленному старту даже самых маленьких пользовательских приложений и примеров. "Холодный старт" приложения, когда все необходимые библиотеки ещё не оказались в системном кэше Linux, занимали около минуты. После долгой загрузки приложения работали без нареканий, но медленный старт стал серьёзной проблемой для прототипирования и разработки. Для сокращения времени первого запуска использовалось tmpfs, средство Linux, позволяющее монтировать файловую систему, которая находится в оперативной памяти. Однако это не решало общей проблемы. После первых итераций разработки, каждая из которых сопровождалась тщательным анализом, из библиотеки были исключены все сторонние библиотеки, некоторая часть функциональности была заменена на стандартные средства .NET, которые бы не требовали загрузки сторонних библиотек и JIT-компиляции внешних иерархий классов, другая была переписана вручную, с использованием лицензий открытого программного обеспечения и декомпиляторов. Самыми затратными сборками оказались Rx[14], библиотека использует небольшое подмножество этого фреймворка, реализованное поверх стандартной библиотек F#, а так же предоставляет Rx-совместимые преобразования. Отказ от библиотек и использование ленивой инициализации позволили серьёзно сократить время запуска, но также не решили общей проблемы. Которая заключалась в том, что JIT-компиляция не слишком хорошо подходит для такого вида устройств.

6.3. AOT

Mono при переходе на мобильные и встраиваемых платформы также сталкнулась с нецелесобразностью использования JIT-компиляции на встраиваемых устройствах в качестве основного типа выполнения. Проблемы, связанные с JIT-компиляцией, стали серьёзно проблемой для Mono не только из-за времени старта. Мобильная платформа iOS из-за соображений безопасности не позволяет динамически генерировать исполняемый код. Для Mono с сущестует AOT-компилятор[2] (Ahead of Time), которые применяется в качестве альтернативы для JIT. При использовании режима AOT-компилятора, байт-код транслируется в машинные инструкции до непосредственного запуска приложения.

К преимуществам данного подхода можно отнести отсутствие ограничений на время кодогенерации, которые являются существенными для JIT-компиляторов. Отсут-

[14]http://www.introtorx.com Lee Campbell

ствие временных ограничений позволяет производить более сложные оптимизации. Но самым большим плюсом является сокращение времени запуска программы, а так же уменьшени е потребления памяти и ресурсов процессора. Однако статическая кодогенерация вынуждает компилятор производить максимально корректный код, в то время как JIT способен производить оптимизации основанные на анализе данных запущенной программы. Такие оптимизации остаются корректными только внутри небольшого временного окна, но позволяют многократно ускорить выполнение.

Mono предоставляет несколько вариантов кодогенерации, которые позволяют сэкономить. Результатом трансляции является динамически подключаемая библиотека, для Linux это ELF[13], которые чаще всего сохраняются с расширением .so. Внутри этого файла находится древовидная структура с кодом от методов.

Использование AOT-компиляции библиотеки, пользовательских программ, а также наиболее часто используемых сборок из GAC, полученных анализом используемых библиотек с помощью переменной окружения для mono "LOG-LEVEL=debug" и ключей -v (verbose) позволил многократно ускорить время работы и запуска.

6.4. Reflection

В языках и средах выполнения с богатыми системами типов, возможностью интроспекции, динамической типизацией и рефлексией, к которым относится .NET, невозможно полностью провести AOT-компиляцию. Через такой мощный механизм как рефлекция программист получает доступ скрытым и даже readonly полям (C# — поля доступные для изменения только в теле конструктора, в CLR — InitOnly), значения которых активно инлайнятся JIT-компилятором. Кроме того пользователь получает возможность создавать экземпляры существующих классов, генерировать новые классы и сборки. Использование рефлексии чрезвычайно мощный аппарат. Нюансы использования рефлексии — предмет для активной дискусси, но в контексте данной работы, важно то, что для такого невозможно сгенерировать код заранее, а значит этот механизм очень дорогой.

Анализ стандартной библиотеки и многих привычных паттернов помогли избежать лишней кодогенерации во время работы. Оказалось, что некоторые методы никак не связанные с Reflection используют этот механизм для совершенно разных задач, помогающих обойти принципы ООП и много другого. В реализации стандартной библиотеки моно, в случае, если value-type используется в качестве ключей в словаре, обычный компаратер для таких чисел будет сгенерирован через reflection. Более известным использованием рефлексии является ключ "%A" для функции printf, который позволяет красиво выводить большую часть различных данных.

Использование этих и многих других эвристик помогло снизить запуск программы с 50 секунд до 6-7. Что ощутимо ускорило и процесс разработки.

7. Необходимые инструментальные средства

Различия архитектуры контроллеров и ПК, а так же род задач, решаемых с помощью встраиваемых устройств, создают определённые особенности для разработки. Поэтому не все существующие инструменты и библиотеки .NET смогут быть повторно использованы для программирования роботов. Не говоря о том, что во фреймворке остаются не реализованными библиотеки, необходимые для разработки роботов, так же как и инструментальные средства для работы с контроллером.

Как и в случае с необходимыми библиотеками за время существования .NET для него было создано множество инструментов, упрощающих разработку, включая несколько IDE, систем сборок, средств отладки и профилирования как производительности, так и потребления памяти, средств автоматизации тестирования и многого другого.

Особенно привлекательным для робототехники и встраиваемых систем является то, что .NET при выполнении использует байт-код, который не зависит от конкретной архитектуры. Поэтому при разработке для робота отпадает необходимость в кросс-компиляции, и вообще хоть в какой-нибудь специальной подготовке программ. Это позволяет использовать полный набор инструментов, применяемых на всех этапах разработки серверных и ПК приложений, и для разработки встраиваемых систем. Таким образом, отличий между проектами нет как на стадии написания кода (статический анализ, авто дополнения, рефакторинги), так и на стадии компиляции (отсутствие кросс-компиляторов, использование билд серверов, Continuous Integration) и даже времени работы (отладка, всевозможные .net фреймворки для тестирования). У Инструментов есть возможность понимать как исходных код программ, так и получившиеся .exe файлы с CLI-кодом и метаданными. Тестирование может проводиться как на контроллере, так и на host-компьютере, где чтение датчиков и передача данных в другие устройства заменены простыми Mock заглушками, как поступают и в обычном тестировании для изоляции некоторых компонентов, тоже самое касается и отладки.

7.1. Особенности разработки роботов

Возможность использовать все эти инструменты делает разработку ПО для роботов практически не отличимой от всех других областей производства ПО. Тем не менее, остается один важный момент, отличающий программирование для встраиваемых устройств от создания программ для обычного ПК. Код, разрабатываемый на одной машине, будет исполняться на совершенно другом компьютере. И, хотя использование байт-кода почти стирает эту границу, а мощность контроллера и проведённые оптимизации почти не накладывают на разработчика ограничений, ему необходимо постоянно обновлять проект на контроллере. С ростом пользовательского проекта

и, как следствие, количества и размера файлов, поддержка согласованности данных между рабочим компьютером и контроллером становится всё сложнее. А тот, факт, что разработчику приходится отвлекаться от разработки, уменьшает преимущество самого подхода.

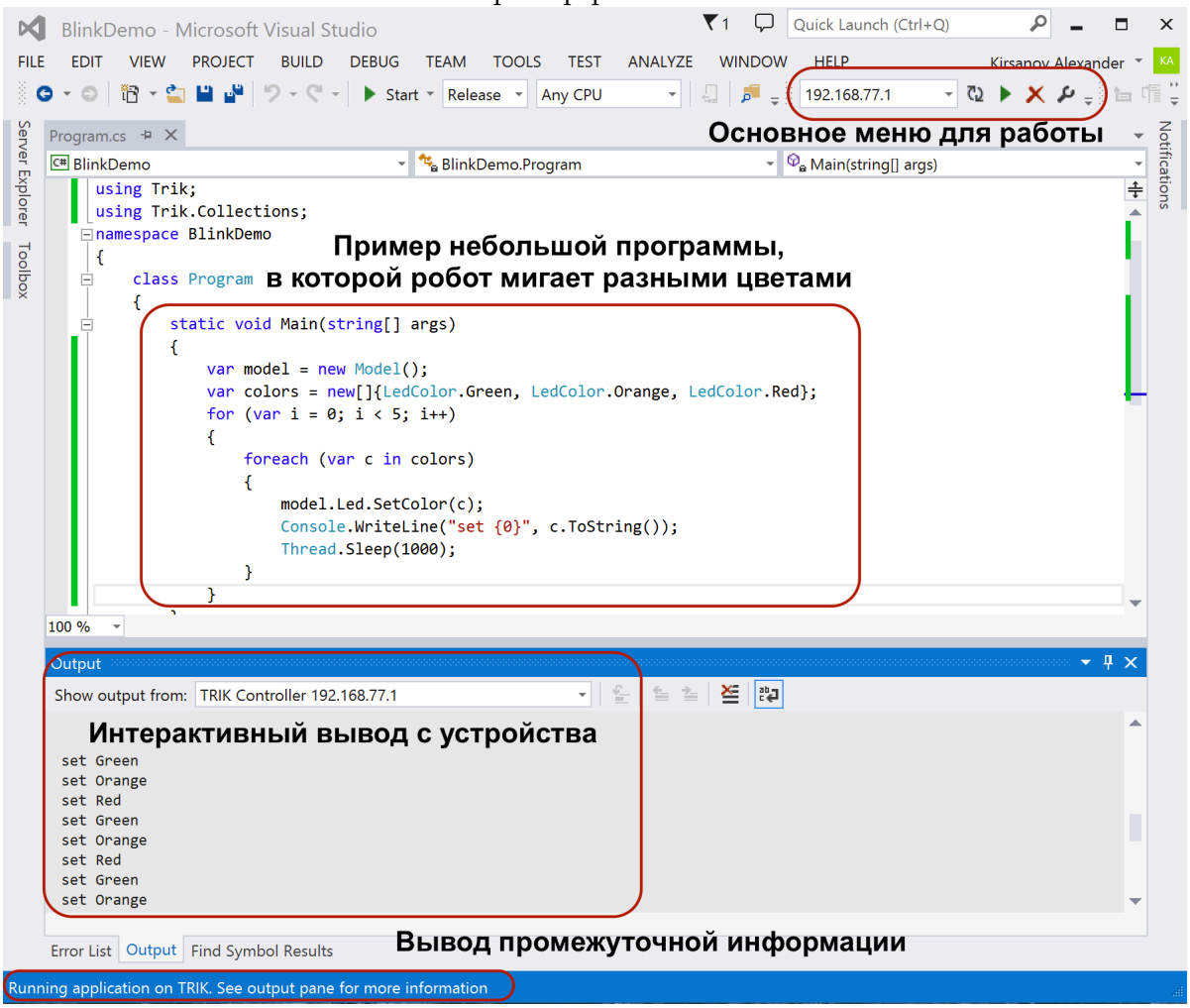
Возникает необходимость в создании средства для автоматической загрузки и запуска приложений. Современные IDE выполняют и более сложные задачи, связанные со сборкой и автоматизацией, поэтому лучшим решением будет не просто создание отдельной программы, отвечающей за своевременное обновление проекта, а интеграци я такой функциональности в среду разработки.

7.2. Расширение для работы с контроллером

Существует несколько популярных IDE, используемых для .NET разработки, самой популярной из которых остается Visual Studio. Интеграция в большой программный продукт всегда сложнее реализации самостоятельной компоненты, но удобство использования, к которому авторы и стремились на протяжении всего проекта, и целостность решения гораздо важнее технических сложностей. Среда разработки успела пережить несколько механизмов для расширения. VS остаётся пронизанной использованием COM-объектов, что заметно усложняет процесс создания расширений. Большим преимуществом системы плагинов для VS является наличие централизованной системы для поиска и установки новых расширений внутри самой Visual Studio.

В ходе работы был создан плагин TRIK-Sharp для автоматизации загрузки и помощи в запуске проектов на контроллере. Все компоненты и кнопки плагина реализованы с помощью средств самой VS и не выделяются из общей функциональности среды разработки. Отображение потока вывода происходит в output pane, как и при работе с обычным консольным приложением. Передача данных и запуск приложения осуществляется через ssh-тоннель, поэтому не привязаны к конкретному контроллеру.

Рис 1. Пример работы плагина



Trik-Upload поддерживает все необходимые функции:

- Автоматическое отслеживание изменений;
- Переключение между несколькими контроллерами;
- Загрузка только измененных частей;
- Работа с несколькими проектами;
- Генерация скрипта для запуска;
- Запуск и остановка приложения;
- Интерактивный вывод потока вывода и ошибок;

8. Результаты и применения

Созданная библиотека и инструментарий дополнили инфраструктуру .NET средствами для разработки роботов. Оптимизации позволили серьёзно сократить потребление ресурсов и увеличить скорость работы. В ходе работы решены все поставленные задачи, а .NET фреймворк может использоваться для высокоуровневой разработки роботов. Исходные коды всех проектов могут быть найдены в соответствующих репозиториях:
`https://github.com/kashmervil/trik-sharp`
`https://github.com/kashmervil/Upload-Extension`

Созданные библиотеки и инструментальные средства помогли сделать программирование роботов практически неотличимым от программирования приложений для ПК. Развитая инструментальная инфраструктура фреймворка, а также архитектурные решения, выбранные при разработке необходимых средств, позволяют в течение минуты превратить Visual Studio в интрумент для разработки роботов. Все компоненты доступны из встроенных пакетных менеджеров для библиотек и расширений. Библиотека и сам подход применялись для обучения в многочисленных летних компьютерных школах и тематических семинарах. Удобство подхода смогли оценить студенты и школьники из разных регионов России. Некоторые из школ:

- AYcamp, Московская область (ABBYY, Яндекс)
- Летняя школа программирования СПбГУ 2014
- Летняя школа-практикум "Компьютерный континуум" (Интел, EMC, JetBrains)
- Весенняя школа программирования СПбГУ (ПМПУ, 7-8 марта 2015)

Различные аспекты настоящей работы послужили материалом для статей и целой серии выступлений. Основные результаты работы изложены в двух статьях, а также нескольких тезисах, некоторые из которых уже опубликованы[8], другие приняты к публикации в материалах конференций CIMSP2015, Современные IT технологии в теории и практике программирования.

Модели роботов, написанные на .NET, использовались для множества демонстраций на различных выставках. А также послужили хорошей основой для создания интерактивных примеров прямо во время выступлений на конференциях Microsoft DevCon 2014, 2015.

Заключение

Современные контроллеры действительно позволяют вести разработку с помощью высокоуровневых языков и языковых платформ. Несмотря на трудности, с которыми пришлось столкнуться при выполнении ВКР, подход повторного использования технологий в робототехнике можно считать успешным. Платформа .NET зарекомендовала себя как удобное и полнофункциональное средство для разработки роботов.

Среди всех реализованных компонентов только библиотека является привязанной к конкретному контроллеру (на данный момент TRIK), а плагин для работы с устройством и проделанные оптимизации могут быть повторно использованы и для других контроллеров. Поэтому в дальнейшей работе планируется реализовать библиотеку для Raspberry Pi.

Список литературы

[1] Cwalina Krzysztof, Abrams Brad. Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries. — 2005.

[2] Documentation Mono Runtime. Ahead of Time Compilation. — 2015.

[3] Foundation The F# Software, contributors. F# Component Design Guidelines, 4 Chaper Guidelines for Libraries for Use from other .NET Languages // fsharp.org. — 2014.

[4] Hu Guoqiang, Tay Wee Peng, Wen Yonggang. Cloud Robotics: Architecture, Challenges and Applications Programming. — Network IEEE. — 2012. — P. 21–28.

[5] International ECMA. Common Language Infrastructure (CLI), Partition III: CIL Instruction Set. — 2012.

[6] J. Richter. CLR via C#. PrePress 4th Edition: Interoperability with Unmanaged Code. — 2012.

[7] Jonas Bonér Dave Farley Roland Kuhn, Thompson Martin. Reactive Manifesto // reactivemanifesto.org. — 2014.

[8] Kirsanov Alexander, Kirilenko Iakov, Melentyev Kirill. Robotics Reactive Programming with F#/Mono // Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia. — CEE-SECR '14. — 2014. — P. 16:1–16:5.

[9] Maja J Mataric Nathan Koenig, Feil-Seifer David. Materials for Enabling Hands-On Robotics and STEM Education. — Interaction Lab.

[10] Managed, Native, and COM Interop Team // http://CodePlex.com/. — 2014.

[11] Petricek Tomas, Syme Don. Collecting Hollywood's garbage: Avoiding space-leaks in composite events // Proceedings of International Symposium on Memory Management. — ISMM 2010.

[12] Skeet Jon. C# in Depth, Chapter 3 Parameterized typing with generics. — 2013. — P. 59 + 105.

[13] (TIS) Tool Interface Standards. Portable Formats Specification, Version 1.1: Executable and Linkable Format (ELF).