Проблематика производительности многоагентных фреймворков для разработки ПО автономных роботов

Михаил Ефремов
Санкт-Петербургский
Электротехнический Университет
СПбГЭТУ "ЛЭТИ"
Email: jakutenshi@gmail.com

***Аннотация*—Данная работа выделяет ряд проблем и задач, выполняемых таким промежуточным ПО, как фреймворки для разработки робототехнических приложений для автономных роботов. Данная статья делает упор на обнаружение критических областей для производительности в многоагентной архитектуре, так как подобный подход является наиболее пригодным для разработки рассматриваемых робототехнических фреймворков. Так же были выделены для исследования ряд существующих и наиболее используемых фреймворков для рассмотрения способов решения проблем разработки такой архитектуры, выделяя подходы и технологии, которые необходимо протестировать для получения различных характеристик производительности распределенной системы.**

I. Введение

Объектом исследования данной статьи являются многоагентные фреймворки для разработки ПО для автономных роботов (далее "фреймворки") с целью установления важных для производительности элементов системы.

Под фреймворками понимается набор промежуточного (middleware) ПО, находящегося по уровню абстракции между ОС и прикладными приложениями, предназначенного для управления неоднородностью аппаратного обеспечения с целью упрощения и снижения стоимости разработки ПО. Кроме того, в состав фреймворков входит набор библиотек и, опционально, инструментов для разработки прикладных программ, обслуживающих систему, в данном случае - автономного робота.

Для программирования роботов доступно множество различных версий фреймворков с различными принципами работы, написанные на разных языках программирования и под разные платформы. В связи с тем, что появляются новые разработки, возникают новые задачи для автономных роботов, а так же технологии разработки ПО для них - возникает желание рассмотреть доступные и развивающиеся в данный момент решения и проанализировать с целью установки характеристик производительности и сравнения по полученным параметрам, чтобы разработчики могли обосновывать свой выбор при разработки приложений для автономных роботов. При этом необходимо учитывать как соответствие фреймворков возможным общим критериям (лицензия, статус разработки), так и важным для конкретной области: разработки ПО для роботов. Для выполнения тестирования, следует определиться с тем, какие задачи, выполняемые фреймворком, являются значимыми для производительности системы в целом.

Целью данной статьи является получения списка проблем и задач, решаемых при разработке фреймворков, которые являются значительными для производительности робототехнической системы в целом.

II. Обзор предметной области

Для обсуждения конкретных решений требуется из всего множества существующих фреймворков выбрать наиболее подходящие для данного исследования.

A. Критерии сравнения аналогов

1) Наличие открытого исходного кода: для лучшего понимания работы фреймворка, а так же лучшего понимания результатов тестирования и возможного улучшения тестовых задач желательно иметь возможность прочитать исходный код реализации функционала, влияющего на результат тестирования. Это может быть, например, реализация коммуникации между приложениями, которые были написаны при помощи исследуемого фреймворка.

2) Наличие документации: написание обоснованных и корректных тестовых задач под конкретный фреймворк очень затруднительно, если отсутствуют инструкций по его использованию. Без наличия документации разработка приложений становится слишком сложной и корректность их выполнения не гарантируется.

3) Текущий статус разработки: проекты, которые больше не поддерживаются разработчиками вполне могут рассматриваться для исследования, но в них скорее-всего используются устаревшие подходы, что приведет к низким показателям производительности.

4) Архитектура фреймворка: разработка робототехнических систем налагает определенные ограничения и требования. Отдельным и самым важным критерием является надежность и устойчивость к ошибкам [1]. Поскольку любой программный продукт, даже хорошо протестированный, содержит какие-либо ошибки, включая те, которые приводят к неожиданному завершению. В случае завершения работы всего программного комплекса из-за ошибки отдельного модуля робот теряет работоспособность. Это особенно важно в таком приложении робототехнических фреймворков, как программирования команд или роя автономных роботов: потеря работоспособности ключевых узлов команды приводит к остановке выполнения поставленной ими задачи. Примером может являться исследование множеством роботов местности или выполняющая задачи команда из разнородных, выполняющих различные роли автономных роботов.

Таким образом, требуется как минимум распределенная архитектура. Чем ближе архитектура будет к P2P (peer-to-peer, одноранговая сеть) и многоагентным системам, тем устойчивее будет все робототехническое ПО. Для данного исследования будут рассматриваться гибридные P2P фреймворки и близкие к чистым P2P.

5) Наличие инструментов для мониторинга и конфигурации системы: для анализа промежуточного ПО и обслуживаемых им приложений требуются такие инструменты, как мониторы используемых ресурсов и системы журналирования. Кроме того, развертывание большого распределенного ПО для проведения тестов является трудоемкой задачей и крайне желательны инструменты конфигурации и развертывания системы на целевой ОС и аппаратном обеспечении.

6) Поддержка различных языков программирования: несмотря на то, что в этой работе наибольший интерес представляет именно промежуточный уровень системы, наличие альтернатив между различными языками программирования прикладного слоя робототехнической системы является преимуществом, поскольку позволяет в зависимости от задачи выбрать между, например, низкоуровневым программированием с возможным преимуществом в производительности и высокоуровневыми языками с наличием удобных для разработки интерфейсов и прикладных библиотек.

В данной работе не столь важны критерии:

- Поддержки ограничений системы реального времени, поскольку это относится не к скорости работы, а к предсказуемости системы. Наличие данного пункта является преимуществом в целом, но относительно производительности оказывается не существенным.
- Поддержка конкретных операционных систем, поскольку рассматривается вопрос производительности слоя абстракции между, собственно, ОС и прикладным ПО.
- Наличие инструментов симуляции, инструментов с графическим пользовательским интерфейсом и набора прикладных библиотек с реализациями наиболее распространенных алгоритмов, поскольку это относится к функциональному уровню системы, ближе к прикладному. Вопросы производительности отдельных инструментов, которые могут требоваться при разработке, отладке и администрированию робототехнических систем не относятся к проблематике потребления ресурсов на поддержание каркаса, основы системы - фреймворка.

B. Представленные для рассмотрения фреймворки

1) ROS: один из наиболее распространенных фреймворков, имеется и обширная документация, и открытый исходный код, разработка продолжается, широко используется. Имеет гибридную архитектуру, средства мониторинга и автоматизации развертывания системы. Для разработки по-умолчанию есть возможность использовать C++ и Python 2.7. [2]

2) MIRA: этот проект активно разрабатывается, имеется открытый исходный код и обширная документация. Имеет децентрализованную архитектуру [3], реализован на C++. Имеются возможности для ведения журналирования приложений, мониторы состояния коммуникаций и ресурсов системы. Для разработки предлагается использовать C++, Python и JavaScript. [4]

3) MOOS: развивающийся проект, имеется документация и исходный код. На данный момент разрабатывается бета-версия MOOS 10. Проблема обеих версий: клиент-серверная архитектура, которая является спорным решением для разработки робототехнических систем из-за проблем устойчивости ПО к ошибкам. По этой причине в данной работе этот фреймворк рассматриваться не будет. [5]

4) OROCOS/Rock: очень распространенный фреймворк, один из немногих поддерживающих ограничения реального времени. Документация обширная, разработка в основном ведется над набором инструментов Rock. Используется гибридная децентрализованная архитектура. Разработка ведется на C++, для разработки прикладных программ предоставляются такие языки, как C++, Python, Simulink [1]. Имеются инструменты для развертывания и мониторинга системы. [6], [7]

5) ASEBA: для программирования используется концепция языков программирования пятого поколения: GUI, блоки, коннекторы. Разработка является скорее обучающим продуктом с коммерческой составляющей в виде конкретной модели робота thymio. Рассматриваться в данной работе не будет. [8]

6) SmartSoft: разрабатывающийся проект с обширной документацией. Для реализации компонентов используется C++. Практически не используется децен-

трализация, основной шаблон взаимодействия клиент-сервер. Кроме того, используется многопоточный подход, а не многопроцессный [9], что ставит под вопрос общую устойчивость всей системы. В данной работе рассматриваться не будет.

7) YARP: активно разрабатывающийся фреймворк с открытым исходным кодом. Имеется обширная и подробная документация. Является одним из немногих практически полностью децентрализованных фреймворков, кроме того, поддерживает ограничения систем реального времени. Разрабатывается в основном на C++ и поддерживает такие языки как C++, Python, Java, Octave. Инструменты мониторинга и развертывания приложений имеются. [10]

8) OpenRTM-aist: распространенный фреймворк с открытым исходным кодом, является реализацией стандарта RT-middleware. Основная проблема: часть документации, при скромном содержании, на японском языке. Архитектура гибридная децентрализованная, имеются инструменты мониторинга, журналирования и развертывания, языки разработки: C++, Java, Python. [11]

9) URBI: данный фреймворк имеет много недостатков: приостановленная разработка, о чем свидетельствует последнее изменение от 2014 года [12] и не работающий сайт самого проекта [13], централизованная архитектура. Рассматриваться в данной работе не будет.

В таблице I отображено соответствие найденных фреймворков предложенным выше критериям.

Таким образом, в исследовании будут учавствовать следующие фреймворки:

- ROS
- MIRA
- OROCOS/Rock
- YARP
- OpenRTM-aist

III. Выбор метода решения

Отобрав фреймворки с децентрализованной структурой, требуется определить те части, которые оказывают наибольшее влияние на производительность. Для этого требуется:

- Разобрать базовые проблемы децентрализованных систем.
- Выявить на какие аспекты производительности каждая из проблем влияет.
- Рассмотреть какие решения были приняты в выбранных для исследования фреймворках.

На основании полученных данных требуется выделить задачи и отслеживаемые характеристики для тестирования фреймворков.

IV. Описание метода решения

A. Проблемы децентрализованных систем

Для распределенной децентрализованной многоагентной системы важны следующие аспекты [1]:

1) Тип архитектуры: большая часть фреймворков использует гибридную архитектуру, поскольку такой подход позволяет избежать ряд проблем, а именно:

- Не требуется инкапсулировать в каждый узел системы общие сервисы, как, например, службу поиска других узлов: эту задачу выполняет специально выделенный узел. Так же поступают и с агенствами - контейнерами, создающие, регистрирующие и хранящие узлы-агенты.
- Уменьшение нагрузки на сеть коммуникаций, поскольку вся нагрузка по общим для всех узлов запросам смещается в сторону обработки на выделенных узлах. Это приводит к появлению критических узлов в системе, в случае отказа которых весь робот может оказаться неспособным продолжать выполнение поставленных задач. Таким образом, в таком подходе требуется отдельно рассматривать производительность таких критических узлов системы, как служба поиска узлов, служба именования и, возможно, другие выделенные сервисы.

В случае, если используется максимально децентрализованная архитектура, то возникают следующие проблемы:

- Увеличивается сложность узлов, а так же нагрузка на обработку данных между ними. Тем не менее то, насколько эффективно используются возможности фреймворка по взаимодействию между узлами, зависит от разработчика на прикладном уровне.
- Увеличивается нагрузка на систему коммуникации фреймворка. Под системой коммуникации понимается набор технологий и протоколов, обеспечивающих обмен данными между узлами системы. Часто для обеспечения коммуникации используются отдельные узлы, создаваемые самим фреймворком, будь то брокеры объектных запросов (ORB) при использовании CORBA или реализации шаблона взаимодействия “издатель-подписчик” при помощи топиков (от англ. topic - тема), которые имеют свою внутреннюю реализацию.

2) Тип взаимодействия между узлами: при использовании многоагентной архитектуры используется несколько разных способов взаимодействия между узлами-агентами:

- Порты, реализующие независимое чтение из входных портов, запись во входные и взаимодействие “один ко многим”.
- Топики, реализующие шаблон “издатель-подписчик” и взаимодействие “многие ко многим”.
- События, реализующие шаблон “наблюдатель” и асинхронное взаимодействие “один ко многим”.

Таблица I
Соответствие найденных робототехнических фреймворков выделенным критериям

НАЗВАНИЕ	ОТКРЫТЫЙ КОД	НАЛИЧИЕ ПОНЯТНОЙ ДОКУМЕНТАЦИИ	ПОСЛЕДНИЕ ИЗМЕНЕНИЯ	АРХИТЕКТУРА	ИНСТРУМЕНТЫ МОНИТОРИНГА	ПОДДЕРЖКА ЯП
ROS	Да	Да	Недавно	Гибридная	Да	C++, Python
MIRA	Да	Да	Недавно	Децентрализованная	Да	C++, Python, JavaScript
MOOS	Да	Да	Недавно	Централизованная	Да	C++, Java
OROCOS/Rock	Да	Да	2016	Гибридная	Да	C++, Python, Simulink
ASEBA	Да	Нет	Недавно	Распределенная	Да	Собственный язык
SmartSoft	Да	Да	Недавно	Распределенная	Да	C++
YARP	Да	Да	Недавно	Децентрализованная	Да	C++, Python, Java, Octave
OpenRTM-aist	Да	Нет	2016	Гибридная	Да	C++, Java, Python
URBI	Да	Нет	2016	Централизованная	Да	C++, Java, urbiscript

Таблица II
Реализация критических для производительности задач для отобранных фреймворков

НАЗВАНИЕ	ЦЕНТРАЛИЗОВАННЫЕ СЕРВИСЫ	ВЗАИМОДЕЙСТВИЕ МЕЖДУ УЗЛАМИ	МЕХАНИЗМЫ КОММУНИКАЦИИ	ТИП СООБЩЕНИЙ	ВЗАИМОДЕЙСТВИЕ С АППАРАТУРОЙ
ROS	Сервисы поиска, именования, сервис параметров	Топики, параметры, сервисы	TCP, UDP, собственный протокол rosserial	Бинарный	Инкапсуляция в узлах
MIRA	Нет	Топики, RPC	Внутрипроцессное взаимодействие, TCP	Бинарный, XML, JSON	Инкапсуляция в узлах и RPC-API
OROCOS/Rock	Сервисы поиска, именования	Порты, сервисы, события, параметры	CORBA, TCP, UDP, SSL,UNIX Sockets, MQueue, EtherCAT, CanOPEN	Сериализация на основе CORBA	Инкапсуляция в узлах и RPC-API
YARP	Сервис имен	Порты, топики	ACE, TCP, UDP, внутрипроцессное взаимодействие	Бинарный	Инкапсуляция в узлах и динамически подключаемые библиотеки
OpenRTM-aist	Сервис имен	Порты, сервисы, параметры	TCP, UDP, SSL,UNIX Sockets, CORBA	Сериализация на основе CORBA	Инкапсуляция в узлах и RPC-API

- Сервисы, предоставляющие реактивный способ поведения узлам: возможность отправлять запрос от узла-клиента на узел-сервис, реализуя взаимодействие выполнения удаленных процедур.
- Свойства, предоставляющие возможность менять состояние узлов при помощи селекторов параметров агента (get/set). Реализация может отличаться в зависимости от архитектуры: гибридная архитектура представляет выделенный узел-сервис - службу свойств, децентрализованная инкапсулирует свойства узлов непосредственно в агентах системы.

Большинство фреймворков дает выбор: какой тип взаимодействия между узлами использовать, но, в случае с реализацией сервисов может возникнуть простой системы до тех пор, пока не работающий или перегруженный обработкой запросов узел, предоставлявший требуемый функционал, не будет восстановлен. Реализация отдельных типов взаимодействий влияет практически на все аспекты производительности: задержку, вычислительные ресурсы, использование памяти, энергопотребление. Для каждого из фреймворков следует тестировать каждый из способов взаимодействия по всем показателям производительности.

3) Механизмы коммуникации: для доставки сообщений обычно используются различные подходы и протоколы. Обычно разработчики пишут свои решения на основе стека TCP/IP с некоторыми оптимизациями,

например реализация узлов как отдельных потоков внутри процесса-агенства. Это позволяет использовать общую память процесса и ускорить взаимодействие между узлами внутри процесса-агенства, которые могут располагаться в пределах одного вычислительного устройства. Это влечет за собой возможность потерять доступ ко всем узлам агенства в случае неисправностей из-за ошибок, допущенных при реализации агентов. Эта проблема решается репликацией агенств на вычислительной системе. Кроме того для доставки сообщений между узлами используются как более высокоуровневые технологии (CORBA, ICE, ACE), так и приближенные к аппаратной части (EtherCAT, I2C, CANBus). Выбор механизма доставки данных между узлами влияет на задержку получения информации, на пропускную способность между узлами, целостность данных, а так же потребление ресурсов вычислительной системы: чем более высокоуровневая технология, тем больше. Дополнительный функционал на уровне коммуникации, например QoS или шифрование так же влияет на производительность. При наличии такого функционала его следует тестировать отдельно.

4) Тип сообщений: сообщения в зависимости от их структуры влияют на производительность. Бинарные типы сообщений имеют меньший объем, а следовательно на их передачу требуется меньше энергии и времени. Структурированные типы, вроде XML и JSON, хорошо поддаются анализу для разработчика, но могут занимать больше времени на обработку своих данных, а так же требуют больше времени на передачу данных между узлами.

5) Способ взаимодействия с аппаратурой: существует два основных способа предоставить интерфейс от аппаратной части системы к прикладному ПО:

- Инкапсулировать взаимодействие с аппаратурой в отдельных узлах, отображающих устройства.
- Использовать промежуточный слой - сервер, который отвечает на запросы узлов и устройств.
- Динамически связывающиеся библиотеки.

От способа обращения будет зависеть отзывчивость системы на внешнее взаимодействие. Кроме того, различные реализации может использовать различное количество ресурсов. Кроме того, в случае использования дополнительного слоя абстракции с клиент-серверным взаимодействием между аппаратным слоем системы и фреймворком является уязвимым подходом с точки зрения устойчивости системы.

B. Решения проблем в различных реализациях фреймворков

На таблице II показано сопоставление возникающих проблемы и их решений для отобранных для исследования фреймворков [1], [3], [14]—[17].

C. Рассматриваемые области тестирования

- Централизованные сервисы, если они имеются во фреймворке с целью установления их быстродействия, устойчивости, потребления ресурсов памяти.
- Способы взаимодействия между узлами с целью установления задержки передачи сообщений между узлами, возможных затрат ресурсов на обработку сообщений.
- Коммуникация между узлами многоагентной системы с целью установления пропускной способности, затрат вычислительных ресурсов на передачу данных. При использовании CORBA и ACE - объем потребляемой памяти на поддержку коммуникации этими методами. При использовании шифрования - задержку передачи сообщения и затраты вычислительных ресурсов на преобразование данных.
- Формат передачи данных между узлами с целью установления скорости сериализации и десериализации, объема передаваемых данных, скорости извлечения данных в случае, если информация сжимается.
- Интерфейсы между аппаратной частью и фреймворком с целью установления задержки реакции системы на окружение.

V. Заключение

В ходе исследования были отобраны 5 фреймворков по различным критериям, среди которых самые важные: открытый исходный код, понятная и доступная документация, децентрализованная или гибридная архитектура.

В ходе исследования многоагентной архитектуры робототехнических фреймворков были выявлены 6 основных источников влияния на производительность данного промежуточного слоя системы:

- Наличие в системе централизованных узлов.
- Типы взаимодействия между узлами системы.
- Механизмы коммуницирования между узлами системы.
- Дополнительный функционал в системе коммуницирования: например шифрование или QoS.
- Типы сообщений, используемых для передачи информации между узлами системы.
- Реализация интерфейса доступа к аппаратной части автономного робота

Исходя из выявленных проблем была составлена таблица решений этих задач отобранными фреймворками, а так же сформулированы основные показатели производительности системы в зависимости от контекста рассматриваемой проблемы.

В данной работе не рассматривалось некоторое количество фреймворков из-за централизованной архитектуры. В теории, опущенные из изучения разработки

могут иметь интерес для тестирования производительности, но из-за разницы архитектур имеется большая разница в возникающих проблемах и задачах. Кроме того было показано, что сильно централизованные архитектуры плохо применимы к разработке робототехнических приложений для автономных роботов.

В дальнейшем, на основе полученной информации, планируется провести тестирование отобранных фреймворков в контексте выявленных важных для производительности областей в многоагентных робототехнических фреймворках.

Список литературы

[1] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz и S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development", *Robotics and Autonomous Systems*, т. 60, № 6, с. 803—821, 2012.

[2] *Documentation - ros wiki*, http://wiki.ros.org/, последний доступ 21 декабря, 2017 года.

[3] E. Einhorn, T. Langner, R. Stricker, C. Martin и H.-M. Gross, "Mira-middleware for robotic applications", в *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, с. 2591—2598.

[4] *Mira: Mira reference documentation*, http://www.mira-project.org/MIRA-doc/index.html, последний доступ 21 декабря, 2017 года.

[5] *Moos : Main - documentation browse*, http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Documentation, последний доступ 21 декабря, 2017 года.

[6] *Main page | the orocos project*, http://www.orocos.org/wiki/main-page, последний доступ 21 декабря, 2017 года.

[7] *Rock the robot construction kit : Table of contents*, https://www.rock-robotics.org/stable/documentation/toc.html/, последний доступ 21 декабря, 2017 года.

[8] *Overview & download - thymio & aseba*, https://www.thymio.org/en:start, последний доступ 21 декабря, 2017 года.

[9] H. Ulm, *Smartsoft approach*, http://www.servicerobotik-ulm.de/drupal/?q=node/19, последний доступ 20 декабря, 2017 года.

[10] *Yarp : Welcome to yarp*, http://www.yarp.it/index.html, последний доступ 21 декабря, 2017 года.

[11] *General information | openrtm-aist*, http://www.openrtm.org/openrtm/en/node/495, последний доступ 21 декабря, 2017 года.

[12] *Github repository aldebaran/urbi: Robotic programming language*, https://github.com/aldebaran/urbi, последний доступ 21 декабря, 2017 года.

[13] *Urbi web-site status: Is comming soon*, http://www.urbiforge.org/, последний доступ 21 декабря, 2017 года.

[14] H. Bruyninckx, "Open robot control software: The orocos project", в *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, IEEE, т. 3, 2001, с. 2523—2528.

[15] G. Metta, P. Fitzpatrick и L. Natale, "Yarp: Yet another robot platform", *International Journal of Advanced Robotic Systems*, т. 3, № 1, с. 8, 2006.

[16] N. Mohamed, J. Al-Jaroodi и I. Jawhar, "Middleware for robotics: A survey", в *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, Ieee, 2008, с. 736—742.

[17] A. Elkady и T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography", *Journal of Robotics*, т. 2012, 2012.