ВикипедиЯ

Модель акторов

Материал из Википедии — свободной энциклопедии

Модель а́кторов — математическая модель параллельных вычислений, строящаяся вокруг понятия «актора» (англ. *actor* — актёр, действующий субъект), считающегося универсальным примитивом параллельного исполнения. Актор в данной модели взаимодействует путём передачи сообщений с другими акторами, в ответ на получаемые сообщения может принимать локальные решения, создавать новые акторы, посылать свои сообщения, устанавливать, как следует реагировать на последующие сообщения.

Создана как теоретическая база для ряда практических реализаций параллельных систем.

Содержание

История

Основные идеи и база для модели заложены в 1973 году публикации Хьюитта, Бишопа и Штайгера[1]. На процесс формирования модели оказали влияние языки программирования Лисп, Симула и ранние версии Smalltalk, а также методы параметрической защиты и коммутации пакетов. Основной мотивацией создания модели стала задача построения распределённых вычислительных систем на базе сотен и тысяч независимых компьютеров, оснащённых собственной локальной памятью и коммуникационными интерфейсами[2]. С появлением многопроцессорных систем и многоядерных архитектур интерес к модели акторов возрос также вне контекста распределённых систем.

В 1975 году разработана операционная семантика для модели акторов[3][4]. В 1977 году выработана система аксиоматических законов для моделей акторов[5]. В 1981 году создана денотационная семантика модели (семантика переходов)[2][6], развитая и обобщённая в 1985 году[7]; в результате этих работ теория моделей акторов признана развитой и проработанной.

В 1990-е годы созданы формализмы, не полностью соответствующие модели акторов (не формализующие гарантированную доставку сообщений), но имеющие практический интерес, в частности, несколько различных алгебр акторов[8][9] и интерпретацию на базе линейной логики[10].

Фундаментальные концепции

По аналогии с философией объектно-ориентированного программирования, где каждый примитив рассматривается как объект, модель акторов выделяет в качестве универсальной сущности понятие «актора». Актор является вычислительной сущностью, которая в ответ на полученное сообщение может одновременно:

- отправить конечное число сообщений другим акторам;
- создать конечное число новых акторов;
- выбрать поведение, которое будет использоваться при обработке следующего полученного сообщения.

Не предполагается существования определённой последовательности вышеописанных действий и все они могут выполняться параллельно.

Отделение отправителя от посланных сообщений стало фундаментальным достижением модели акторов: тем самым обеспечивается асинхронная связь и управление структурами в виде формы передачи сообщений[11].

Получатели сообщений идентифицируются по адресу, который иногда называют «почтовым адресом». Таким образом, актор может взаимодействовать только с теми акторами, адреса которых он имеет, может извлечь адреса из полученных сообщений или знать их заранее, если актор создан им самим.

Модель характеризуется внутренне присущим параллелизмом вычислений внутри одного актора и между акторами, динамическим созданием акторов, включением адресов акторов в сообщения, а также взаимодействием только через прямой асинхронный обмен сообщениями без каких-либо ограничений на порядок прибытия сообщений.

Применения

Модель акторов может использоваться в качестве основы для моделирования, понимания и аргументации по широкому спектру параллельных систем, например:

- электронная почта (e-mail) может быть смоделирована как система акторов: клиенты моделируются как акторы, а адреса электронной почты — как адреса акторов;
- веб-сервисы с конечными точками SOAP могут быть смоделированы как адреса акторов;
- объекты с семафорами (например, в Java и C#) могут быть смоделированы как параллельно-последовательный преобразователь, при условии, что их реализация такова, что сообщения могут приходить постоянно (возможно, они хранятся во внутренней очереди). Параллельно-последовательный преобразователь является важным видом актора, характеризующийся тем, что он постоянно доступен для прихода новых сообщений. Каждое сообщение, отправленное на параллельно-последовательный преобразователь, гарантированно будет получено;
- язык программирования SmallTalk[12] построен исключительно на взаимодействии объектов посредством сообщений друг другу. Код каждого объекта при этом исполняется изолированно от соседей в параллельной среде.
- нотация тестирования и управления тестами (как TTCN-2, так и TTCN-3) довольно близко соответствует модели акторов — в TTCN актором является тест компонента: либо параллельный тест компонента (PTC), либо главный тест компонента (MTC); тесты компонентов могут отправлять и получать сообщения на/от удалённых партнеров (равноправные тесты компонентов или тест интерфейса системы), причём последний идентифицируется по его адресу; каждый тест компонента имеет дерево поведения, связанное с ним; тесты компонентов запускаются параллельно, и могут быть динамически созданы родительскими тестами компонентов; встроенные языковые конструкции позволяют определить действия, которые необходимо выполнить, когда сообщение получено из внутренней очереди сообщений, а также отправить сообщения другим равноправным субъектом или создать новые тесты компонентов.

Семантика передачи сообщений

Неограниченные недетерминированные разногласия

Возможно, первыми параллельными программами были обработчики прерываний. В процессе работы, как правило, компьютеру необходимо реагировать на внешние события, которые могут происходить в заранее неизвестный момент времени (асинхронно по отношению к выполняющейся в данный момент программе) — например, получать информацию извне (символы с клавиатуры, пакеты из сети и так далее). Наиболее эффективно обработка таких событий реализуется с помощью так называемых прерываний. Когда происходит событие, выполнение текущей программы «прерывается», и запускается обработчик прерывания, который выполняет действия, необходимые для реагирования на событие (например, получает

поступающую информацию и сохраняет её в буфер, откуда она может быть впоследствии считана), после чего основная программа продолжает свою работу с того места, где она была прервана.

В начале 1960-х годов прерывания стали использовать для имитации одновременного выполнения нескольких программ на одном процессоре[13]. Наличие параллелизма с общей памятью привело к проблеме управления параллелизмом. Первоначально эта задача задумывалась как один из мьютексов на отдельном компьютере. Эдсгер Дейкстра разработал семафоры, а позднее, в период между 1971 и 1973 годами, Чарльз Хоар и Пер Хансен для решения проблемы мьютексов разработали мониторы[14][15][16]. Однако, ни одно из этих решений не создавало в языках программирования конструкций, которые бы инкапсулировали доступ к совместным ресурсам. Инкапсуляцию сделали позже Хьюитт и Аткинсон, построив параллельно-последовательный преобразователь ([Hewitt, Atkinson 1977, 1979] и [Atkinson 1980]).

Первые модели вычислений (например, машина Тьюринга, машина Поста, лямбда-исчисление и тому подобные) были основаны на математике и использовали понятие глобального состояния, чтобы определить «шаг вычисления» (позднее эти понятия обобщены в работах Маккарти и Дейкстры[17][18]). Каждый шаг вычисления шёл от одного глобального состояния вычислений до следующего. Глобальный подход к состоянию был продолжен в теории автоматов для конечных автоматов и машин со стеком, в том числе их недетерминированные версии. Такие недетерминированные автоматы имеют свойство ограниченного недетерминизма. То есть, если машина всегда стоит перед тем, как она переходит в исходное состояние, то имеется ограничение на число состояний, в которых она может находиться.

Дейкстра развил дальше подход с недетерминированными глобальными состояниями. Модель Дейкстры породила споры о неограниченном недетерминизме — свойстве параллельных вычислений, при котором величина задержки в обслуживании запроса может стать неограниченной в результате арбитражного соперничества за общие ресурсы, в то же время гарантируется, что запрос в конечном итоге будет обслужен. Хьюитт утверждал, что модель акторов должна обеспечить гарантии на предоставление услуги. Хотя в модели Дейкстры не может быть неограниченного количества времени между выполнением последовательных операций на компьютере, параллельно выполняемая программа, которая начала свою работу в строго определённом состоянии, может быть прервана лишь в ограниченном числе состояний[18]. Следовательно, модель Дейкстры не может обеспечить гарантии предоставления услуги. Дейкстра утверждал, что невозможно осуществить неограниченный недетерминизм.

Хьюитт утверждал иное: не существует ограничения на время, которое затрачивается на работу участка вычислений, называемого арбитром для урегулирования конфликтов. Арбитры имеют дело с разрешением таких ситуаций. Часы компьютера работают асинхронно с внешними входами: вводом с клавиатуры, доступом к диску, сетевым входом и так далее. Так что для получения сообщения, отправленного на компьютер, может пройти неограниченное время, и за это время компьютер может пройти через неограниченное количество состояний.

Неограниченный недетерминизм является характерной чертой модели акторов, в которой используется математическая модель Клингера, основанная на теории областей[2]. В модели акторов не существует глобального состояния.

Прямая связь и асинхронность

Сообщения в модели акторов не обязательно буферизуются. В этом её резкое различие с предшествующими подходами к модели одновременных вычислений. Отсутствие буферизации вызвало большое недоразумение во время развития модели акторов, и до сих пор эта тема является предметом споров. Некоторые исследователи утверждают, что сообщения буферизуются в «эфире» или «окружающей среде». Кроме того, сообщения в модели акторов просто посылаются (например, пакеты в IP). Никаких требований на синхронное рукопожатие с получателем не существует.

Создание новых акторов и передача адресов в сообщениях означают изменяемую топологию

Естественным развитием модели акторов была возможность передачи адресов в сообщениях. Под влиянием сетей с коммутацией пакетов Хьюитт предложил разработать новую модель одновременных вычислений, в которой связь не будет иметь вообще никаких обязательных полей, все они могут быть пустыми. Конечно, если отправитель сообщения желает, чтобы получатель имел доступ к адресам, которых он ещё не имеет, адрес должен быть отправлен в сообщении.

В процессе вычислений, возможно, потребуется отправить сообщение получателю, от которого позже нужно получить ответ. Способ сделать это состоит в том, чтобы отправить сообщение, в котором записан адрес другого актора, называемого *возобновлением* (иногда его также называют *продолжением* или *стеком вызовов*). Получатель может затем создать ответное сообщение, которое будет отправлено на *возобновление*.

Создание акторов плюс включение адресов участников в сообщения означает, что модель акторов имеет потенциально переменную топологию в своих отношениях друг с другом, походя на объекты в языке Симула, которые в своих отношениях друг с другом также имеют переменную топологию.

По сути одновременно

В отличие от предыдущего подхода, основанного на комбинировании последовательных процессов, модель акторов была разработана как одновременная модель по своей сути. Как написано в теории моделей акторов, последовательность в ней представляет собой особый случай, вытекающий из одновременных вычислений.

Никаких требований о порядке поступления сообщений

Хьюитт был против включения требований о том, что сообщения должны прибывать в том порядке, в котором они отправлены на модель актора. Если желательно упорядочить входящие сообщения, то это можно смоделировать с помощью очереди акторов, которая обеспечивает такую функциональность. Такие очереди акторов упорядочивали бы поступающие сообщения так, чтобы они были получены в порядке FIFO. В общем же случае, если актор `X` отправляет сообщение `M1` актору `Y`, а затем тот же актор `X` отправляет другое сообщение `M2` к `Y`, то не существует никаких требований о том, что `M1` придёт к `Y` раньше `M2`.

В этом отношении модель акторов зеркально отражает систему коммутации пакетов, которая не гарантирует, что пакеты будут получены в порядке отправления. Отсутствие гарантий порядка доставки сообщений позволяет системе коммутации пакетов буферизовать пакеты, использовать несколько путей отправки пакетов, повторно пересылать повреждённые пакеты и использовать другие методы оптимизации.

Например, акторы могут использовать конвейер обработки сообщений. Это означает, что в процессе обработки сообщения `M1` актор может варьировать поведение, которое будет использоваться для обработки следующего сообщения. В частности, это означает, что он может начать обработку ещё одного сообщения `M2` до завершения обработки `M1`. На том основании, что актору предоставлено право использования конвейера обработки сообщений, ещё не означает, что он этот конвейер *обязан* использовать. Будет ли сообщение конвейеризовано или нет — относится к задачам технического компромисса. Как внешний наблюдатель может узнать, что обработка сообщения актора прошла через конвейер? На этот счёт не существует никакой двусмысленности в отношении применения актором возможности конвейеризации. Только если в конкретной реализации выполнение конвейерной оптимизации сделано неправильно, может произойти не ожидаемое поведение, а нечто другое.

Локальность

Другой важной характеристикой модели акторов является локальность: при обработке сообщения актор может отправлять сообщения только по тем адресам, которые он получил из этого сообщения, по адресам, которые он уже имел до получения сообщения, и по адресам, которые он создал при обработке сообщения.

Локальность также означает, что не может одновременно произойти несколько изменений адресов. В этом отношении модель акторов отличается от некоторых других моделей параллелизма, например, от сетей Петри, в которых реализации одновременно могут быть удалены из нескольких позиций и размещены по другим адресам.

Композиция систем акторов

Идея композиции систем акторов в более крупные образования является важным аспектом модульности, которая была разработана в докторской диссертации Гуля Ага[7], позже развитой им же вместе с Ианом Мейсоном, Скоттом Смитом и Каролин Талкотт[4].

Поведение

Основным новшеством модели акторов было введение понятия поведения, определённого как математическая функция, выражающая действия актора, когда он обрабатывает сообщения, включая определение нового поведения на обработку следующего поступившего сообщения. Поведение обеспечивает функционирование математической модели параллелизма.

Поведение также освобождает модель акторов от деталей реализации, как, например, в Smalltalk-72 это делает маркер интерпретатора потока. Однако, важно понимать, что эффективное внедрение систем, описываемых моделью акторов, требует расширенную оптимизацию.

Моделирование других параллельных систем

Другие системы параллелизма (например, исчисление процессов) могут быть смоделированы в модели акторов с использованием двухфазного протокола фиксации[19].

Теорема вычислительных представлений

В модели акторов существует *теорема вычислительных представлений* для замкнутых систем, в том смысле, что они не получают сообщений извне. В математической записи замкнутая система, обозначаемая как S, строится как наилучшее приближение для начального поведения, называемого $\bot_S$, с использованием аппроксимирующей функции поведения $\mathbf{progression}_S$, построенной для S следующим образом (согласно публикации Хьюитта 2008 года):

$$\mathbf{Denote}_S \equiv \sqcup_{i \in \omega} \mathbf{progression}_S^i(\bot_S)$$

Таким образом, S может быть математически охарактеризована в терминах всех его возможных поведений (в том числе с учётом неограниченного недетерминизма). Хотя Denote_S не является реализацией S, она может быть использована для доказательства следующего обобщения тезиса Чёрча — Тьюринга[20]: если примитив актора замкнутой системы акторов являются эффективным, то его возможные выходы рекурсивно перечислимы. Доказательство непосредственно вытекает из теоремы вычислительных представлений.

Связь с математической логикой

Развитие модели акторов имеет интересную связь с математической логикой. Одной из ключевых мотиваций для её развития была необходимость управления аспектами, которые возникли в процессе развития языка программирования Planner. После того как модель акторов была первоначально сформулирована, стало важно определить мощность модели в отношении тезиса Роберта Ковальского о том, что «вычисления могут быть сгруппированы по логическим выводам». Тезис Ковальского оказался ложным для одновременных вычислений в модели акторов. Этот результат всё ещё является спорным и противоречит некоторым предыдущим представлениям, поскольку тезис Ковальского верен для последовательных вычислений и даже для некоторых видов параллельных вычислений, например, для лямбда-исчислений.

Тем не менее были предприняты попытки расширения логического программирования для одновременных вычислений. Однако, Хьюитт и Ага в работе 1999 года утверждают, что результирующая система не является дедуктивной в следующем смысле: вычислительные шаги параллельных систем программирования логики не следуют дедуктивно из предыдущих шагов.

Миграция

Миграцией в модели акторов называется способность актора изменить своё местоположение. Например, Аки Йонезава в своей диссертации моделировал почтовую службу, в которой акторы-клиенты могли войти, изменить местоположение во время работы и выйти. Актор, который мог мигрировать, моделировался как актор с определённым местом, изменяющимся при миграции актора. Однако достоверность этого моделирования является спорной и служит

предметом исследований.

Безопасность

Безопасность акторов может быть обеспечена одним из следующих способов:

- аппаратным управлением, к которому акторы подключены физически;
- через специальное оборудование, как, например, в Барроуз B5000, Лисп-машине и так далее;
- через виртуальную машину, как, например, в виртуальной машине Java, в в виртуальной машине CLR и так далее;
- через операционную систему, как, например, в системах с параметрической защитой;
- использованием электронной цифровой подписи и/или шифрования для акторов и их адресов.

Синтез адресов акторов

Тонким моментом в модели акторов является возможность синтеза адреса актора. В некоторых случаях система безопасности может запрещать синтез адресов. Поскольку адрес актора — это просто битовая строка, то, очевидно, его можно синтезировать, хотя, если битовая строка достаточно длинная, подобрать адрес актора довольно трудно или даже невозможно. SOAP в качестве адреса конечной точки использует URL, по которому актор расположен. Так как URL является строкой символов, то, очевидно, её можно синтезировать, хотя если применить шифрование, то подобрать строку практически невозможно.

Синтезирование адресов акторов обычно моделируется с помощью отображения. Идея состоит в использовании системы акторов для выполнения отображения на фактические адреса акторов. Например, структура памяти компьютера может быть смоделирована как система акторов, которая даёт отображение. В случае адресов SOAP это моделирование DNS и отображение URL.

Отличие от других моделей параллельной передачи сообщений

Первая из опубликованных работ Робина Милнера о параллелизме[21] была примечательна тем, что не была основана на композиции последовательных процессов, отличалась от модели акторов, потому что она была основана на фиксированном количестве процессов фиксированного числа связей в топологии строк, используемых для синхронизации связи. Оригинальная модель взаимодействующих последовательных процессов (CSP), опубликованная Энтони Хоаром[22], отличается от модели акторов, потому что основана на параллельной композиции фиксированного числа последовательных процессов, связанных в фиксированную топологию и общающихся с помощью синхронной передачи сообщений на основе имён процессов. Более поздние версии CSP отказались от связи на основе имён процессов, приняв принцип анонимной связи по каналам. Этот подход используется также в работе Милнера об исчислении общающихся систем и пи-исчислении.

Этим обеим ранним моделям Милнера и Хоара свойствен ограниченный недетерминизм.

Современные теоретические модели взаимодействующих систем[23] прямо предусматривают неограниченный недетерминизм.

Актуальность

Через сорок лет после публикации закона Мура продолжающееся возрастание производительности микросхем происходит благодаря методам локального и глобального массового параллелизма. Локальный параллелизм задействован в новых микросхемах для 64-разрядных многоядерных микропроцессоров, в мульти-чиповых модулях и высокопроизводительных системах связи. Глобальный параллелизм в настоящее время задействован в новом оборудовании для проводной и беспроводной широкополосной пакетной коммутации сообщений. Ёмкость хранения за счёт как локального, так и глобального параллелизма, растёт в геометрической прогрессии.

Модель нацелена на решение следующих задач построения вычислительных систем:

- масштабируемость: проблема расширения параллелизма, как локального, так и нелокального;
- прозрачность: преодоление пропасти между локальным и нелокальным параллелизмом. Прозрачность в настоящее время является спорным вопросом. Некоторые исследователи выступают за строгое разделение между локальным параллелизмом, используемом в языках параллельного программирования (например, Java и C#), и нелокальным параллелизмом, используемом в SOAP для веб-сервисов. Строгое разделение приводит к отсутствию прозрачности, что вызывает проблемы, когда желательно/необходимо внести изменения в локальные и нелокальные методы доступа к веб-службам;
- противоречивость: противоречивость является нормой, потому что все очень большие системы знаний о взаимодействии информационных систем человечества противоречивы. Эта противоречивость распространяется на документацию и технические характеристики очень больших систем (например, программное обеспечение Microsoft Windows, и так далее), которые являются внутренне противоречивыми.

Многие идеи, введённые в модели акторов, в настоящее время находят также применение в многоагентных системах по этим же причинам[24]. Ключевым отличием является то, что агент системы (в большинстве определений) накладывает дополнительные ограничения на акторов, как правило, требуя, чтобы они использовали обязательства и цели.

Модель акторов применяется также в клиентах облачных вычислений[25].

Программирование с акторами

Среди ранних языков программирования с поддержкой акторов — Act 1, 2 и 3[26][27], Acttalk[28], Ani[29], Cantor[30], Rosette[31]

Более поздние языки, ориентированные на модель акторов: Actor-Based Concurrent Language (ABCL), ActorScript, AmbientTalk[32], Axum[33]. Среди языков программирования общего назначения, где используется понятие актора — E, Elixir[34], Erlang, Io, SALSA[35], Scala[36][37].

Разработаны библиотеки и табличные структуры с акторами для обеспечения

актороподобного стиля программирования на языках, которые не имеют встроенных акторов.

Библиотеки и табличные структуры с акторами			
Название	**Дата последнего выпуска**	**Лицензия**	**Языки программирования**
ActiveJava	2008	?	Java
Actor	2013-05-31	MIT	Java
Actor-CPP	2012-03-10[38]	GPL 2.0	C++
Actor Framework	2013-11-13	Apache 2.0	.NET
ActorKit	2011-09-13[39]	BSD	Objective-C
Akka	2015-04-23	Apache 2.0	Java and Scala
Akka.NET	2016-01-18	Apache 2.0	.NET
C++ Actor Framework (CAF)	2015-11-25[40]	Boost Software License 1.0 and BSD 3-Clause	C++11
Celluloid	2016-01-19[41]	MIT	Ruby
Cloud Haskell	2015-06-17[42]	BSD	Haskell
CloudI	2015-12-24[43]	BSD	C/C++, Elixir/Erlang/LFE, Java, Javascript, Perl, PHP, Python, Ruby
Functional Java	2016-02-15[44]	BSD	Java
GPars	2014-05-09[45]	Apache 2.0	Groovy
Jetlang	2013-05-30[46]	New BSD	Java
Korus	2010-02-04	GPL 3	Java
Kilim[47]	2011-10-13[48]	MIT	Java
LabVIEW Actor Framework	2012-03-01[49]	?	LabVIEW
libprocess	2013-06-19	Apache 2.0	C++
NAct	2012-02-28	LGPL 3.0	.NET
OOSMOS	2016-02-17[50]	GPL 2.0 и коммерческая	C, C++
Orbit	2016-02-16[51]	New BSD	Java
Orleans	2019-06-04[52]	MIT	.NET
Panini	2014-05-22	MPL 1.1	Собственный язык программирования
Peernetic	2007-06-29	LGPL 3.0	Java
PostSharp	2014-09-24	Коммерческая / Freemium	.NET
Pulsar	2016-11-24[53]	New BSD	Python
Pulsar	2016-02-18[54]	LGPL/Eclipse	Clojure

Библиотеки и табличные структуры с акторами			
Название	**Дата последнего выпуска**	**Лицензия**	**Языки программирования**
Pykka	2019-12-02[55]	Apache 2.0	Python
Remact.Net	?	MIT	.NET
Retlang	2011-05-18[56]	New BSD	.NET
S4	2012-07-31[57]	Apache 2.0	Java
SObjectizer	2016-02-11	New BSD	C++11
Termite Scheme	2009-05-21	LGPL	Scheme
Theron	2014-01-18[58]	MIT[59]	C++
Thespian	2019-09-11[60]	GoDaddy Public Release[61]	Python
QP	2015-09-29[62]	GPL 2.0 и коммерческая	C and C++
Quasar	2016-01-18[63]	LGPL/Eclipse	Java

Примечания

↑ Показывать компактно

1. Карл Хьюитт, Питер Бишоп, Ричард Штайгер: Универсальный модульный формализм акторов для искусственного интеллекта. IJCAI, 1973 (англ.)
2. Уильям Клингер, Основы семантики акторов. MIT, докторская диссертация по математике, июнь 1981 (https://dspace.mit.edu/handle/1721.1/6935) (англ.)
3. [Ирен Грейф, Семантика коммуникативных параллельных процессов. MIT, докторская диссертация, август 1975] (англ.)
4. Г. Ага, И. Мейсон, С. Смит, К. Талкотт. Основания для вычислений акторов. Journal of Functional Programming, январь, 1993 (англ.)
5. Г. Бейкер, К. Хьюитт. Законы взаимодействующих параллельных процессов. IFIP, август 1977 года (англ.)
6. Карл Хьюитт. Что такое обязательство? Физическое, организационное и социальное. (http://www.pcs.usp.br/~coin-aamas06/10_commitment-43_16pages.pdf) (англ.)
7. Гуль Ага, Акторы: Модель параллельных вычислений в распределенных системах. MIT Press, докторская диссертация, 1986 (https://dspace.mit.edu/handle/1721.1/6952) (англ.)
8. M. Gaspari, G. Zavattaro. An Algebra of Actors. Technical Report UBLCS-97-4. University of Bologna, 1997
9. G. Agha, P. Thati. An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language. (http://formal.cs.uiuc.edu/papers/ATactors_festschrift.pdf) (недоступная ссылка). Дата обращения 14 февраля 2011. Архивировано (https://web.archive.org/web/20040420064252/http://formal.cs.uiuc.edu/papers/ATactors_festschrift.pdf) 20 апреля 2004 года.

10. *John Darlington; Y. K. Guo.* Formalizing Actors in Linear Logic (неопр.). — International Conference on Object-Oriented Information Systems, 1994.
11. Carl Hewitt. *Viewing Control Structures as Patterns of Passing Messages* Journal of Artificial Intelligence. June 1977.
12. Среда исполнения SmallTalk как современный пример реализации (сайт проекта Pharo) (https://pharo.org/).
13. П. Хансен. Истоки параллельного программирования: от семафоров к удалённому вызову процедур. Springer, 2002 (англ.)
14. Per Hansen, *Monitors and Concurrent Pascal: A Personal History*, Comm. ACM 1996, pp 121—172
15. Hansen, P., *Operating System Principles*, Prentice-Hall, July 1973.
16. C.A.R. Hoare, *Monitors: An Operating System Structuring Concept*, Comm. ACM Vol. 17, No. 10. October 1974, pp. 549—557
17. [McCarthy and Hayes 1969]
18. [Dijkstra 1976]
19. Frederick Knabe. A Distributed Protocol for Channel-Based Communication with Choice PARLE 1992.
20. Клини, 1943
21. Robin Milner. Processes: A Mathematical Model of Computing Agents in Logic Colloquium 1973.
22. C.A.R. Hoare. Communicating sequential processes (http://portal.acm.org/citation.cfm?id=359585&dl=GUIDE&coll=GUIDE&CFID=19884966&CFTOKEN=55490895) CACM. August 1978.
23. [Hoare 1985], [Roscoe 2005]
24. Хьюитт, 2006b, 2007b
25. Карл Хьюитт. Организация масштабируемых, надёжных, конфиденциальных клиентов для облачных вычислений. IEEE Internet Computing, v. 12 (5), 2008 (англ.)
26. Генри Либерман. Обзор Act 1. MIT AI, июнь 1981 (ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-625.pdf) (недоступная ссылка) (англ.)
27. Генри Либерман. Мышление о многом сразу без путаницы: Параллелизм в Act 1. MIT AI, июнь 1981 (ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-626.pdf) (недоступная ссылка) (англ.)
28. Jean-Pierre Briot. Acttalk: A framework for object-oriented concurrent programming-design and experience 2nd France-Japan workshop. 1999.
29. Ken Kahn. A Computational Theory of Animation MIT EECS Doctoral Dissertation. August 1979.
30. William Athas and Nanette Boden Cantor: An Actor Programming System for Scientific Computing in Proceedings of the NSF Workshop on Object-Based Concurrent Programming. 1988. Special Issue of SIGPLAN Notices.
31. Darrell Woelk. Developing InfoSleuth Agents Using Rosette: An Actor Based Language Proceedings of the CIKM '95 Workshop on Intelligent Information Agents. 1995.
32. Dedecker J., Van Cutsem T., Mostinckx S., D'Hondt T., De Meuter W. Ambient-oriented Programming in AmbientTalk. In «Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP), Dave Thomas (Ed.), Lecture Notes in Computer Science Vol. 4067, pp. 230—254, Springer-Verlag.», 2006

33. Microsoft Cooking Up New Parallel Programming Language — Application Development — News & Reviews — eWeek.com (http://www.eweek.com/c/a/Application-Development/Microsoft-Cooking-Up-New-Parallel-Programming-Language-Axum-868670/)
34. *Dave Thomas.* Chapter 14. Working with Multiple Processes // Programming Elixir. — Pragmatic Bookshelf, 2014. — 280 p. — ISBN 978-1-937785-58-1.
35. Carlos Varela and Gul Agha. Programming Dynamically Reconfigurable Open Systems with SALSA. ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings, 2001
36. Philipp Haller and Martin Odersky, Event-Based Programming without Inversion of Control, Proc. JMLC, September, 2006 (http://lampwww.epfl.ch/~odersky/papers/jmlc06.pdf)
37. Philipp Haller and Martin Odersky, Actors that Unify Threads and Events. Technical report LAMP, January, 2007 (http://lamp.epfl.ch/~phaller/doc/haller07coord.pdf) (недоступная ссылка). Дата обращения 14 февраля 2011. Архивировано (https://web.archive.org/web/20110607225711/http://lamp.epfl.ch/~phaller/doc/haller07coord.pdf) 7 июня 2011 года.
38. Changes - actor-cpp - An implementation of the actor model for C++ - Google Project Hosting (https://code.google.com/p/actor-cpp/source/list). Code.google.com. Дата обращения 25 февраля 2016.
39. Commit History · stevedekorte/ActorKit · GitHub (https://github.com/stevedekorte/ActorKit/commits/master). Github.com. Дата обращения 25 февраля 2016.
40. Tags · actor-framework/actor-framework · GitHub (https://github.com/actor-framework/actor-framework/tags). Github.com. Дата обращения 25 февраля 2016.
41. celluloid | RubyGems.org | your community gem host (http://rubygems.org/gems/celluloid). RubyGems.org. Дата обращения 25 февраля 2016.
42. Cloud Haskell: Erlang-style concurrent and distributed programming in Haskell (https://haskell-distributed.github.io/). Github.com. Дата обращения 25 февраля 2016.
43. CloudI Dowanloads (https://sourceforge.net/projects/cloudi/files/?source=navbar). sourceforge.net. Дата обращения 25 февраля 2016.
44. Functional Java Releases (https://github.com/functionaljava/functionaljava/tree/master/etc/release-notes). GitHub. Дата обращения 25 февраля 2016. (недоступная ссылка)
45. GPars Releases (https://github.com/GPars/GPars/releases). GitHub. Дата обращения 25 февраля 2016.
46. jetlang downloads (https://code.google.com/archive/p/jetlang/downloads). Code.google.com. Дата обращения 25 февраля 2016.
47. *Srinivasan, Sriram* (2008). "Kilim: Isolation-Typed Actors for Java (http://www.malhar.net/sriram/kilim/kilim_ecoop08.pdf) " (PDF). *European Conference on Object Oriented Programming ECOOP 2008*. Проверено 2016-02-25.
48. Commit History · kilim/kilim · GitHub (https://github.com/kilim/kilim/commits/master). Github.com. Дата обращения 25 февраля 2016.
49. Community: Actor Framework, LV 2011 revision (version 3.0.7) (https://decibel.ni.com/content/docs/DOC-18308). Decibel.ni.com. Дата обращения 25 февраля 2016.

50. OOSMOS Version History (http://www.oosmos.com/version-history) (недоступная ссылка). OOSMOS. Дата обращения 25 февраля 2016. Архивировано (https://web.archive.org/web/20160310151340/http://oosmos.com/version-history) 10 марта 2016 года.
51. Orbit, GitHub, tag 0.7.1 release (https://github.com/electronicarts/orbit/releases/tag/v0.7.1). GitHub. Дата обращения 25 февраля 2016. (недоступная ссылка)
52. Orleans, GitHub, tag 2.3.4 release (https://github.com/dotnet/orleans/releases/tag/v2.3.4). GitHub. Дата обращения 4 июня 2019.
53. Pulsar Release Notes (https://quantmind.github.io/pulsar/history/1.6.html).
54. Pulsar on GitHub (https://github.com/puniverse/pulsar).
55. Changes — Pykka 2.0.2 documentation (https://www.pykka.org/en/latest/changes/). pykka.org. Дата обращения 2020-27-01.
56. Changes - retlang - Message based concurrency in .NET - Google Project Hosting (https://code.google.com/p/retlang/source/list). Code.google.com. Дата обращения 25 февраля 2016.
57. Commit History · s4/s4 · Apache (https://git1-us-west.apache.org/repos/asf?p=incubator-s4.git) (недоступная ссылка). apache.org. Дата обращения 25 февраля 2016. Архивировано (https://web.archive.org/web/20160306073515/https://git1-us-west.apache.org/repos/asf?p=incubator-s4.git) 6 марта 2016 года.
58. Theron - Version 6.00.02 released (http://www.theron-library.com/index.php?t=news) (недоступная ссылка). Theron-library.com. Дата обращения 25 февраля 2016. Архивировано (https://web.archive.org/web/20160316122155/http://www.theron-library.com/index.php?t=news) 16 марта 2016 года.
59. Theron (http://www.theron-library.com/index.php?t=page&p=license) (недоступная ссылка). Theron-library.com. Дата обращения 25 февраля 2016. Архивировано (https://web.archive.org/web/20160304000109/http://www.theron-library.com/index.php?t=page&p=license) 4 марта 2016 года.
60. Thespian release history on PyPI (https://pypi.org/project/thespian/#history).
61. Thespian Releases (https://godaddy.github.io/Thespian/doc/releases.html). godaddy.com. Дата обращения 29 сентября 2015.
62. QP Active Object Frameworks - Browse Files at (https://sourceforge.net/projects/qpc/files/). Sourceforge.net. Дата обращения 25 февраля 2016.
63. Quasar GitHub (https://github.com/puniverse/quasar).

Литература

- *John C. Mitchell.* Concepts in programming languages. — Cambridge University Press, 2003. — 529 p. — ISBN 978-0-521-78098-8.
- *И. Федотов.* Модели параллельного программирования. — М.: Солон-Пресс, 2012. — С. 384. — ISBN 978-5-91359-102-9.
