# Модель акторов

Материал из Википедии — свободной энциклопедии [править | править вики-текст]

Текущая версия страницы пока не проверялась опытными участниками и может значительно отличаться от версии, проверенной 7 июля 2016; проверки требуют 2 правки.

В компьютерных науках **моде́ль а́кторов** представляет собой математическую модель параллельных вычислений, которая трактует понятие «актор» как универсальный примитив параллельного численного расчёта: в ответ на получаемые сообщения актор может принимать локальные решения, создавать новые акторы, посылать свои сообщения, а также устанавливать, как следует реагировать на последующие сообщения. Модель акторов возникла в 1973 году[1], использовалась как основа для понимания исчисления процессов и как теоретическая база для ряда практических реализаций параллельных систем.

**Содержание** [скрыть]

## История [ править | править вики-текст ]

В отличие от предыдущих моделей вычислений, появление модели акторов было стимулировано физикой, в том числе общей теорией относительности и квантовой механикой. На процесс формирования модели оказали также влияние языки программирования Lisp, Симула и ранние версии Smalltalk, а также методы параметрической защиты и коммутации пакетов. Развитие модели было «мотивировано перспективой высокопараллельных вычислительных машин, состоящих из десятков, сотен и даже тысяч независимых микропроцессоров, каждый со своей собственной локальной памятью и коммуникационным процессором, общающихся через сеть высокопроизводительной связи»[2]. С массовым распространением параллелизма, возникшего благодаря развитию многоядерных архитектур, интерес к модели акторов значительно возрос.

После публикации Хьюитта, Бишопа и Штайгера в 1973 году Ирен Грейф разработала операционную семантику для модели акторов как часть своей докторской диссертации[3]. Два года спустя Генри Бейкер и Хьюитт опубликовали множество аксиоматических законов для систем акторов[4]. Другие значимые вехи включают диссертацию Уильяма Клингера 1981 года[2], в которой представлена денотативная семантика, основанная на мощности доменов, и диссертацию Гуля Ага 1985 года, в которой дано дальнейшее развитие семантической модели Клингера[5]. В результате этих работ теория модели акторов получила полное развитие.

## Фундаментальные концепции [ править | править вики-текст ]

Модель акторов исходит из такой философии, что *всё вокруг является акторами*. Это похоже на философию объектно-ориентированного программирования, где *всё вокруг является некоторыми объектами*, но отличается тем, что в объектно-ориентированном программировании программы, как правило, выполняются последовательно, в то время как в модели акторов вычисления по своей сути совпадают по времени.

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.

Может существовать произвольная последовательность вышеописанных действий, и все они могут выполняться параллельно.

Развязка отправителя и посланных сообщений стала фундаментальным достижением модели акторов, обеспечившая асинхронную связь и управление структурами как прототип передачи сообщений[6].

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 акторов 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.

## Формальные системы [ править | править вики-текст ]

Для описания модели акторов за прошедшие годы были разработаны несколько различных формальных систем. Например:

- операционная семантика[3][7];
- законы для систем акторов[4];
- денотационная семантика[en][2][8];
- семантика переходов[5].

Существуют также формализмы, не полностью соответствующие модели акторов в том аспекте, что они не формализуют гарантированную доставку сообщений. К ним, в частности, относятся:

- несколько различных алгебр акторов[9][10];
- линейная логика[11].

## Применения [ править | править вики-текст ]

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;
- 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.

## Предшествующие модели [ править | править вики-текст ]

Модель акторов сформировалась на базе предшествующих моделей вычислений.

### Лямбда-исчисление [ править | править вики-текст ]

Лямбда-исчисление Алонзо Чёрча можно рассматривать как самый первый язык программирования обмена сообщениями[1] (см. также Абельсон и Суссман 1985). Например, нижеприведённое лямбда-выражение реализует древовидную структуру данных, если используется с параметрами leftSubTree и rightSubTree. Если на входе такого дерева дать в качестве параметра сообщение "getLeft", оно возвратит leftSubTree, а если дать сообщение "getRight", то возвратится rightSubTree.

```
λ(leftSubTree,rightSubTree)
  λ(message)
    if (message == "getLeft") then leftSubTree
    else if (message == "getRight") then rightSubTree
```

Семантика лямбда-исчислений выражается при помощи подстановок переменных, в которых значения параметров заменяются в теле вызываемых лямбда-выражений. Модель подстановок непригодна для параллелизма, поскольку она не обеспечивает возможность совместного использования ресурсов. Под влиянием лямбда-исчислений интерпретатор языка программирования Lisp использует структуры данных, называемые средой (environment), такие, что значения параметров не должны заменяться в теле запускаемых лямбда-выражений. Это обеспечивает совместное использование эффектов обновления общих структур данных, но не обеспечивает параллелизм.

### Симула [ править | править вики-текст ]

Язык Симула был пионером в использовании передачи сообщений для вычислений, связанных с дискретными приложениями моделирования событий. В предыдущих языках моделирования эти приложения были громоздкими и немодульными. На каждом шаге времени нужно было выполнять большую центральную программу и обновлять состояния каждого моделируемого объекта, зависящие от состояния других объектов, с которыми данный объект взаимодействовал на текущем шаге моделирования. Кристен Нюгорд и Оле-Йохан Даль первыми развили идею (впервые изложена на семинаре IFIP в 1967 году) применения методов, встроенных в каждый объект, которые обновляют свои собственные состояния на основании сообщений от других объектов. Кроме того, они ввели структуры классов для объектов с наследованием. Их нововведения значительно повысили модульность программ.

Однако в Симуле вместо истинного параллелизма использовались сопрограммы управления структурами.

### Smalltalk [ править | править вики-текст ]

При разработке Smalltalk-71 Алан Кэй находился под влиянием возможности передачи сообщений в управляемых шаблонами вызовах языка Planner. Хьюитт был заинтригован языком Smalltalk-71, но отложил его применение из-за сложности коммуникаций, которые включают вызовы со многими полями, включая *global*, *sender*, *receiver*, *reply-style*, *status*, *reply*, *operator selector* и т. д.

В 1972 году Кэй посетил MIT и обсудил некоторые свои идеи для Smalltalk-72, основанные на возможностях языка программирования Лого Сеймура Паперта и на модели вычислений «маленький человек», используемой для обучения детей программированию. Однако, передача сообщений в Smalltalk-72 была довольно сложной. Код на языке рассматривался интерпретатором просто как поток символов. Как позже писал Дэн Инголс:

> *Первый встретившийся в динамическом контексте символ в программе использовался для определения получателя последующего сообщения. Поиск имени начинался с типа словаря в*

*текущей активации. В случае ошибки сообщение перенаправлялось создателю активации, и так далее по цепочке, вплоть до отправителя сообщения. Когда связывание, наконец, находило символ, его значение присваивалось получателю нового сообщения, а интерпретатор активировал код для класса этого объекта*.

Таким образом, передача сообщений в Smalltalk-72 была тесно привязана к конкретной модели машины и к синтаксису языка программирования, который не был приспособлен для параллелизма. Кроме того, хотя система загружалась сама, языковые конструкции не были формально определены как объекты, которые отвечают на **Eval** сообщения (см. обсуждение ниже). Это позволило некоторым сделать вывод, что новая математическая модель параллельных вычислений на основе передачи сообщений должна быть проще, чем Smalltalk-72.

Последующие версии языка Smalltalk в значительной степени развивались по пути использования виртуальных методов из языка Симула в структурах программ передачи сообщений. Однако в Smalltalk-72 в объектах появились примитивы, таких как целые числа, числа с плавающей точкой и т. д. Авторы языка Симула рассмотрели принятие таких примитивов в объектах, но воздержались от этого, в основном по соображениям эффективности. В языке Java сначала посчитали целесообразным использовать как примитивы, так и версии объектов целых чисел, чисел с плавающей точкой и др. В языке программирования C# (и более поздних версиях Java, начиная с Java 1.5) принято менее элегантное решение использования *упаковки* и *распаковки*, которые ранее использовались в некоторых реализациях Lisp.

Система Smalltalk впоследствии стала очень популярной и влиятельной, оказав инновационное воздействие на растровые дисплеи, персональные компьютеры, интерфейс браузеров и на многое другое[12]. Тем временем усилия разработчиков модели акторов в MTI сосредоточились на развитии научных и технических основ более высокого уровня параллелизма.

### Сети Петри [ править | править вики-текст ]

До появления модели акторов сети Петри широко использовались для моделирования недетерминированных вычислений. Однако, было признано, что они имеют важное ограничение: они моделируют управление потоком, но не сам поток данных. Следовательно, они не были легко компонуемыми, что тем самым ограничивало их модульность. Хьюитт отметил ещё одну проблему сетей Петри: сложность одновременных действий. Элементарный шаг вычислений в сети Петри представляет собой переход, при котором токены одновременно исчезают на входах перехода и появляются на его выходах. Физические основы использования примитивов с такого рода одновременностью оказались сомнительными. Несмотря на эти очевидные трудности, метод сетей Петри остаётся популярным подходом к моделированию параллелизма и всё еще является предметом активных исследований.

### Нити, блокировки и буферы (каналы) [ править | править вики-текст ]

До модели акторов параллелизм был определён в аппаратных терминах низкого уровня через нити, блокировки и буферы (каналы). Не случайно, конечно, то, что реализация модели акторов обычно использует эти аппаратные возможности. Однако, нет никаких оснований считать, что модель не может быть реализована непосредственно в аппаратуре без эквивалентных аппаратных нитей и блокировок. Кроме того, нет обязательной связи между акторами, нитями и блокировками, которые могут быть вовлечены в вычисления. Реализации модели акторов не связаны непосредственно с нитями и блокировками во всех случаях, совместимых с законами для акторов.

## Семантика передачи сообщений [ править | править вики-текст ]

О семантике передаваемых сообщений в модели акторов.

## Неограниченные недетерминированные разногласия [ править | править вики-текст ]

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 т. д.). 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[*источник не указан 255 дней*].

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 т. д.) были основаны на математике и использовали понятие глобального состояния, чтобы определить *шаг вычисления* (позднее эти понятия обобщены в работах [McCarthy and Hayes 1969] и [Dijkstra 1976]). 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 индетерминизма. 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.

Эдсгер Дейкстра развил дальше подход с недетерминированными глобальными состояниями. Модель Дейкстры породила споры о *неограниченном индетерминизме*. Неограниченный индетерминизм (называемый также *неограниченным недетерминизмом*) является свойством одновременных 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 утверждает, 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 [Dijkstra 1976]. Следовательно, модель Дейкстры не может обеспечить гарантии предоставления услуги. Дейкстра утверждал, что невозможно осуществить неограниченный индетерминизм.

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 т. д. Так что для получения сообщения, отправленного

на компьютер, может пройти неограниченное время, и за это время компьютер может пройти через неограниченное количество состояний.

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

### Прямая связь и асинхронность [ править | править вики-текст ]

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 [ править | править вики-текст ]

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 [ править | править вики-текст ]

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 [ править | править вики-текст ]

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 [ править | править вики-текст ]

Другой важной характеристикой модели акторов является *локальность*. Локальность означает, что 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 [ править | править вики-текст ]

Идея композиции систем акторов в более крупные образования является важным аспектом модульности, которая была разработана в докторской диссертации Гуля Ага[5], позже развитой им же вместе с Ианом Мейсоном, Скоттом Смитом и Каролин Талкотт[7].

### Поведение [ править | править вики-текст ]

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 [ править | править вики-текст ]

Другие системы параллелизма (например, исчисление процессов) могут быть смоделированы в модели акторов с использованием двухфазного протокола фиксации[17].

### Теорема вычислительных представлений [ править | править вики-текст ]

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 индетерминизма). Хотя $\mathrm{Denote}_S$ не является реализацией $S$, она может быть использована для доказательства обобщения тезиса Чёрча-Тьюринга-Россера-Клини (см. Клини, 1943):

*Теорема перечислимости*: 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 [ править | править вики-текст ]

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 [ править | править вики-текст ]

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, изменяющемся при миграции актора. Однако достоверность этого моделирования является спорной и служит предметом исследований.

## Безопасность [ править | править вики-текст ]

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 т. д.;
- через виртуальную машину, как, например, в виртуальной машине Java, в в виртуальной машине CLR и т. д.;
- 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 [ править | править вики-текст ]

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 моделируются 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

[ править | править вики-текст ]

Первая из опубликованных работ Робина Милнера о параллелизме[18] 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[19], 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 индетерминизм. Современные теоретические CSP ([Hoare 1985] и [Roscoe 2005]) прямо предусматривают неограниченный индетерминизм.

## Актуальность [ править | править вики-текст ]

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 (см. Wi-Fi и Ultra Wideband). Ёмкость хранения за счёт как локального, так и глобального параллелизма, растёт в геометрической прогрессии.

По словам Хьюитта (см. Carl Hewitt, 2006a), модель акторов ставит вопросы в области компьютеров и архитектуры связи, языков параллельного программирования и веб-сервисов, включая следующие:

- 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 т. д.), которые являются внутренне противоречивыми.

Многие идеи, введённые в модели акторов, в настоящее время находят также применение в многоагентных системах по этим же причинам (см. Хьюитт, 2006b, 2007b). 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[20].

## Программирование с акторами [ править | править вики-текст ]

**Список примеров в этой статье или её разделе не основывается на авторитетных источниках непосредственно о предмете статьи или её раздела.**
Добавьте ссылки на источники, предметом рассмотрения которых является тема настоящей статьи (или раздела) в целом, и содержащие данные элементы списка как примеры. В противном случае раздел может быть удалён.

Большое число различных языков программирования используют модель акторов или её варианты. Среди них:

### Ранние программные языки с акторами [ править | править вики-текст ]

- Act 1, 2 и 3 [21][22]
- Acttalk [23]
- Ani [24]
- Cantor [25]
- Rosette[26]

### Более поздние программные языки с акторами [ править | править вики-текст ]

- Actor-Based Concurrent Language (ABCL)
- ActorScript
- AmbientTalk[27]
- Axum (язык программирования)[28]
- C#[29]
- E (язык программирования)
- Elixir[30]
- Erlang
- Io

- Ptolemy Project
- Rebeca Modeling Language
- Reia (язык программирования)
- SALSA (язык программирования)[31]
- Scala[32][33]

### Библиотеки и табличные структуры с акторами [ править | править вики-текст ]

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** [показать]

## Литература [ править | править вики-текст ]


- *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.[59]


Более подробный список литературы приведен в викиучебнике .

## См. также [ править | править вики-текст ]

- Многоагентная система

## Примечания [ править | править вики-текст ]

↑
Показывать
компактно

1. ↑ 1 2 Карл Хьюитт, Питер Бишоп, Ричард Штайгер: Универсальный модульный формализм акторов для искусственного интеллекта. IJCAI, 1973 (англ.)
2. ↑ 1 2 3 4 Уильям Клингер, Основы семантики акторов. MIT, докторская диссертация по математике, июнь 1981 (англ.)
3. ↑ 1 2 [Ирен Грейф, Семантика коммуникативных параллельных процессов. MIT, докторская диссертация, август 1975] (англ.)
4. ↑ 1 2 Г. Бейкер, К. Хьюитт. Законы взаимодействующих параллельных процессов. IFIP, август 1977 года (англ.)
5. ↑ 1 2 3 Гуль Ага, Акторы: Модель параллельных вычислений в распределенных системах. MIT Press, докторская диссертация, 1986 (англ.)
6. ↑ Carl Hewitt. *Viewing Control Structures as Patterns of Passing Messages* Journal of Artificial Intelligence. June 1977.
7. ↑ 1 2 Г. Ага, И. Мейсон, С. Смит, К. Талкотт. Основания для вычислений акторов. Journal of Functional Programming, январь, 1993 (англ.)
8. ↑ Карл Хьюитт. Что такое обязательство? Физическое, организационное и социальное. (англ.)
9. ↑ M. Gaspari, G. Zavattaro. An Algebra of Actors. Technical Report UBLCS-97-4. University of Bologna, 1997
10. ↑ G. Agha, P. Thati. An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language.
11. ↑ John Darlington; Y. K. Guo (1994). «Formalizing Actors in Linear Logic» (International Conference on Object-Oriented Information Systems).
12. ↑ Алан Кэй. Ранняя история Smalltalk. ACM SIGPLAN, v. 28 (3), March, 1993, pp. 69–75 (англ.)
13. ↑ П. Хансен. Истоки параллельного программирования: от семафоров к удаленному 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. ↑ Frederick Knabe. A Distributed Protocol for Channel-Based Communication with Choice PARLE 1992.
18. ↑ Robin Milner. Processes: A Mathematical Model of Computing Agents in Logic Colloquium 1973.
19. ↑ C.A.R. Hoare. Communicating sequential processes CACM. August 1978.
20. ↑ Карл Хьюитт. Организация масштабируемых, надёжных, конфиденциальных клиентов для облачных вычислений. IEEE Internet Computing, v. 12 (5), 2008 (англ.)
21. ↑ Генри Либерман. Обзор Act 1. MIT AI, июнь 1981 (англ.)
22. ↑ Генри Либерман. Мышление о многом сразу без путаницы: Параллелизм в Act 1. MIT AI, июнь 1981 (англ.)
23. ↑ Jean-Pierre Briot. Acttalk: A framework for object-oriented concurrent programming-design and experience 2nd France-Japan workshop. 1999.
24. ↑ Ken Kahn. A Computational Theory of Animation MIT EECS Doctoral Dissertation. August 1979.
25. ↑ 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.
26. ↑ Darrell Woelk. Developing InfoSleuth Agents Using Rosette: An Actor Based Language Proceedings of the CIKM '95 Workshop on Intelligent Information Agents. 1995.
27. ↑ 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
28. ↑ Microsoft Cooking Up New Parallel Programming Language - Application Development - News & Reviews - eWeek.com
29. ↑ Akka.NET is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on .NET & Mono
30. ↑ *Dave Thomas.* Chapter 14. Working with Multiple Processes // Programming Elixir. — Pragmatic Bookshelf, 2014. — 280 p. — ISBN 978-1-937785-58-1.
31. ↑ Carlos Varela and Gul Agha. Programming Dynamically Reconfigurable Open Systems with SALSA. ACM SIGPLAN Notices. OOPSLA’2001 Intriguing Technology Track Proceedings, 2001
32. ↑ Philipp Haller and Martin Odersky, Event-Based Programming without Inversion of Control, Proc. JMLC, September, 2006
33. ↑ Philipp Haller and Martin Odersky, Actors that Unify Threads and Events. Technical report LAMP, January, 2007
34. ↑ Changes - actor-cpp - An implementation of the actor model for C++ - Google Project Hosting . Code.google.com. Проверено 25 февраля 2016.
35. ↑ Commit History · stevedekorte/ActorKit · GitHub . Github.com. Проверено 25 февраля 2016.
36. ↑ Tags · actor-framework/actor-framework · GitHub . Github.com. Проверено 25 февраля 2016.
37. ↑ celluloid | RubyGems.org | your community gem host . RubyGems.org. Проверено 25 февраля 2016.
38. ↑ Cloud Haskell: Erlang-style concurrent and distributed programming in Haskell . Github.com. Проверено 25 февраля 2016.
39. ↑ CloudI Dowanloads . sourceforge.net. Проверено 25 февраля 2016.
40. ↑ Functional Java Releases . GitHub. Проверено 25 февраля 2016.
41. ↑ GPars Releases . GitHub. Проверено 25 февраля 2016.
42. ↑ jetlang downloads . Code.google.com. Проверено 25 февраля 2016.
43. ↑ *Srinivasan, Sriram* (2008). "Kilim: Isolation-Typed Actors for Java " (PDF). *European Conference on Object Oriented Programming ECOOP 2008*. Проверено 2016-02-25.
44. ↑ Commit History · kilim/kilim · GitHub . Github.com. Проверено 25 февраля 2016.
45. ↑ Community: Actor Framework, LV 2011 revision (version 3.0.7) . Decibel.ni.com. Проверено 25 февраля 2016.
46. ↑ OOSMOS Version History . OOSMOS. Проверено 25 февраля 2016.
47. ↑ Orbit, GitHub, tag 0.7.1 release . GitHub. Проверено 25 февраля 2016.
48. ↑ Orleans, GitHub, tag 1.1.2 release . GitHub. Проверено 25 февраля 2016.

49. ↑ Pulsar oldchangelog .
50. ↑ Pulsar on GitHub .
51. ↑ Changes — Pykka 1.2.1 documentation . pykka.org. Проверено 25 февраля 2016.
52. ↑ Changes - retlang - Message based concurrency in .NET - Google Project Hosting . Code.google.com. Проверено 25 февраля 2016.
53. ↑ Commit History · s4/s4 · Apache . apache.org. Проверено 25 февраля 2016.
54. ↑ Theron - Version 6.00.02 released . Theron-library.com. Проверено 25 февраля 2016.
55. ↑ Theron . Theron-library.com. Проверено 25 февраля 2016.
56. ↑ Thespian Releases . godaddy.com. Проверено 29 сентября 2015.
57. ↑ QP Active Object Frameworks - Browse Files at . Sourceforge.net. Проверено 25 февраля 2016.
58. ↑ Quasar GitHub .
59. ↑ Модели параллельного программирования (ozon.ru)