О возможности использования среды Ptolemy II в учебном процессе

Доленко Д.В.
Сениченков Ю.Б.

Знакомство с Ptolemy II (1)

- Домашняя страница Ptolemy II: https://ptolemy.berkeley.edu/ptolemyII/index.htm
- «Ptolemy II - это **программный фреймворк** с открытым исходным кодом, поддерживающий эксперименты с **акторно-ориентированным дизайном**. **Акторы (actors)** - это программные компоненты, которые выполняются одновременно и взаимодействуют посредством сообщений, отправляемых через взаимосвязанные порты. Модель - это иерархическая взаимосвязь акторов. В Ptolemy II семантика модели определяется не фреймворком, а скорее программным компонентом модели, называемым **директором (director)**, который реализует **модель вычислений (model of computation)…** Каждый уровень иерархии в модели может иметь своего собственного директора, и отдельные директора могут быть составлены иерархически.»

Что значат эти понятия?

программный фреймворк

акторно-ориентированный дизайн

actor

director

модель вычислений

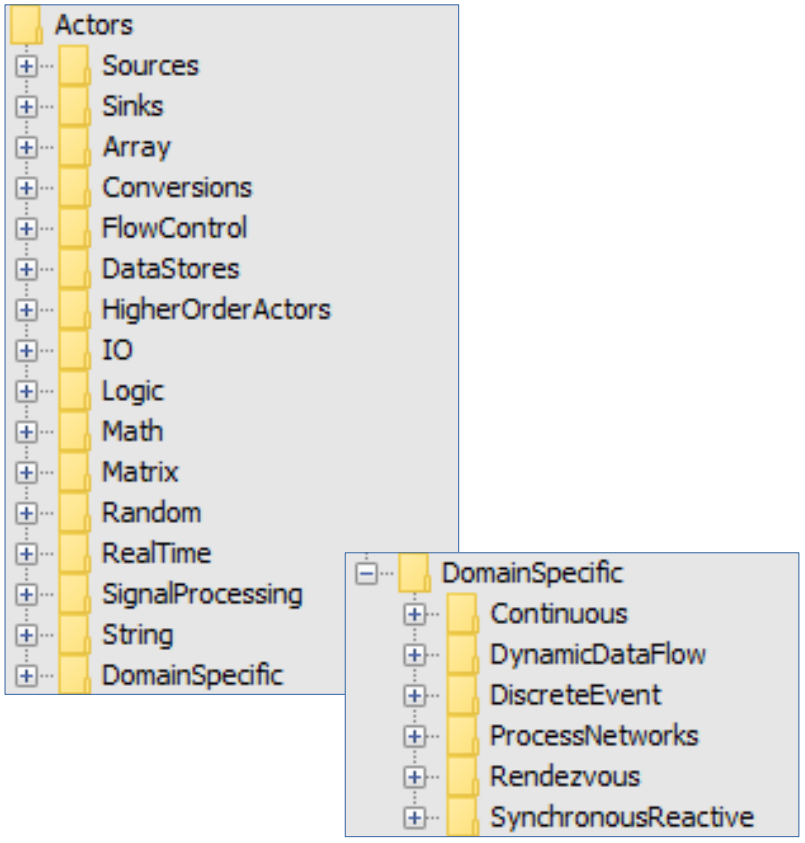
«Ptolemy II - это программный фреймворк с открытым исходным кодом...»

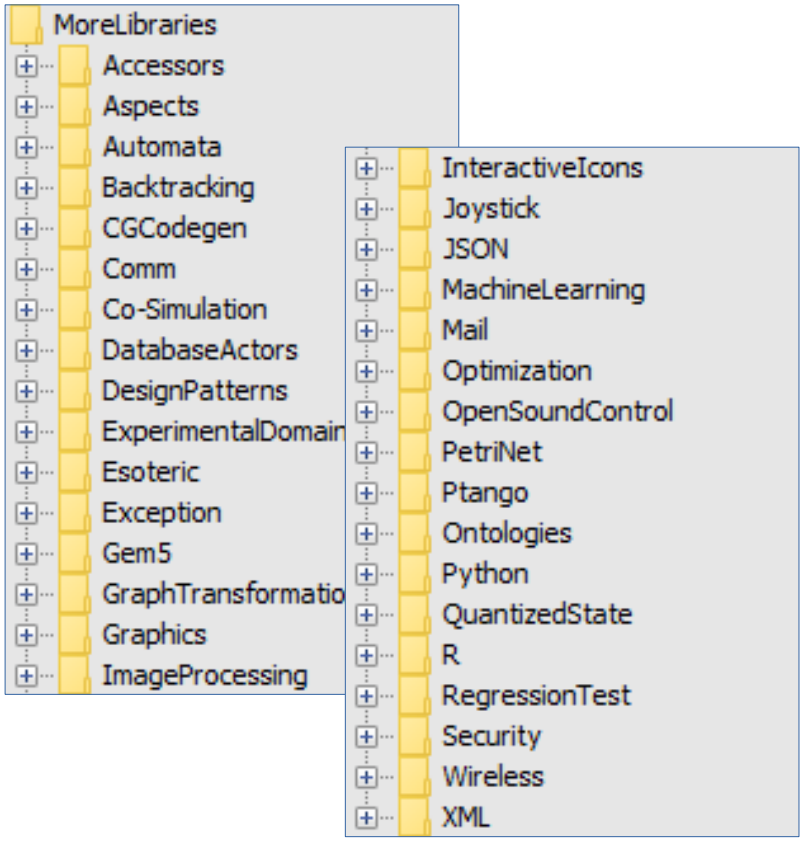
- **Фреймворк** (англицизм от framework — каркас, рама, структура) - программная платформа, определяющая структуру программной системы. Это лишь означает, что в Ptolemy II, как, например, в Simulink или AnyDynamics, можно составить структурную схему из объектов и соединений, сформировав модель.
- Открытый исходный код: https://github.com/icyphy/ptII

«...поддерживающий эксперименты с акторно-ориентированным дизайном»

- «Ptolemy II основан на классе моделей, называемых **акторно-ориентированными моделями**, или **моделями акторов**.»
- **Акторы** (от англ. actor «актёр; действующий субъект») - это компоненты, которые выполняются одновременно и обмениваются данными друг с другом, отправляя сообщения через порты. Это аналог активного объекта UML, но с ограничениями на тип взаимодействия между объектами.
- **Акторно-ориентированный дизайн** — подход к моделированию, в котором акцент ставится на моделировании систем как набора независимых компонентов, которые взаимодействуют друг с другом путём обмена сообщениями.
- Возможность экспериментирования с данным дизайном — возможность экспериментирования с реализованными *моделями вычислений* (различными типами *взаимодействия* акторов друг с другом) и их комбинациями.

Библиотека акторов Ptolemy II





«В Ptolemy II семантика модели определяется... компонентом модели, называемым директором (director), который реализует модель вычислений.»

- Предметная область определяет "законы физики" для взаимодействия между компонентами. Она предоставляет правила, которые управляют одновременным выполнением компонентов и обменом данными между компонентами. Набор таких правил называется **моделью вычислений (model of computation)**.
- Эти правила делятся на три категории:
  - Первый набор правил определяет, что представляет собой компонент. В Ptolemy II под компонентом понимается актор.
  - Второй набор правил определяет механизмы выполнения и параллелизма. Они отвечают на вопросы: «Вызываются ли акторы по порядку?», «Вызываются ли одновременно?», «Вызываются ли недетерминированно?».
  - Третий определяет механизмы коммуникации. «Как акторы обмениваются данными?»
- **Директор** (от лат. dirigere, directum — от di и regere — «управлять») - реализация конкретной модели вычислений.

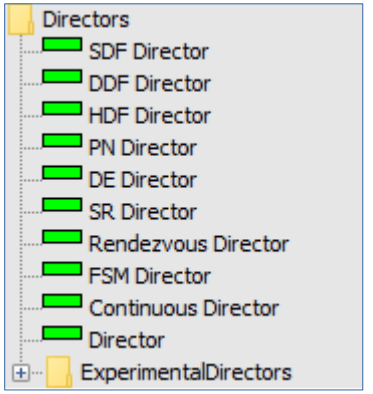
Директор	Механизмы выполнения и параллелизма	Механизмы коммуникации
Synchoronous (Static) Dataflow	«Вызываются по порядку». Порядок выполнения акторов статичен и определяется на стадии «компиляции» модели.	Акторы имеют буфер на входном порту. Генерируемые сообщения от других акторов сохраняются в этом буфере, и при вызове актора из него читается порция данных.
Continuous	«Вызываются одновременно». В момент времени, выбранный решателем, запускаются все акторы.	При запуске актор передаёт значения переменных через порты напрямую акторам назначения.
Rendezvous	«Вызываются недетерминированно». Директор помещает каждый актор в свой собственный поток. Недетерминизм может возникнуть, когда один актор хочет устроить рандеву с *несколькими* другими акторами.	Связь между акторами осуществляется посредством рандеву: когда актор готов послать сообщение, он блокируется до тех пор, пока принимающий актор не будет готов принять его. Аналогично, если актор готов принять сообщение через входной порт, он блокируется до тех пор, пока отправляющий актор не будет готов отправить его.
Discrete-Events	Актор вызывается только в том случае, если у него на входном порту есть сообщение (событие) или он сам ранее запросил у управляющего компонента запуск.	Каждое взаимодействие называется событием и концептуально понимается как мгновенное сообщение, отправляемое одним актором другому. (Аналогично с Continuous директором, но акторы запускаются только по приходу события.)

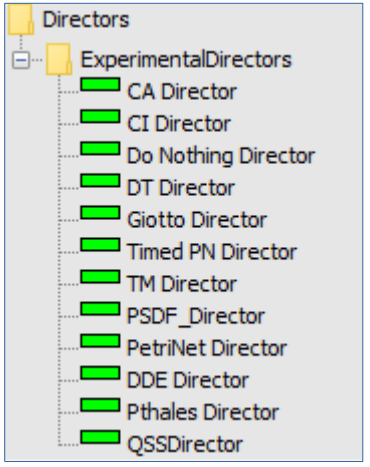
Реализованные в Ptolemy II модели вычислений

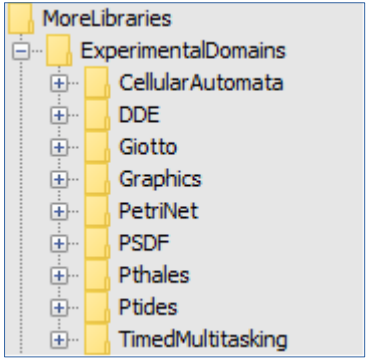
Директора с хорошо формализованной семантикой в Ptolemy II:

- *process networks* (сеть процессов),
- *discrete-events* (дискретные события),
- *dataflow* (синхронные и динамические потоки данных),
- *synchronous/reactive* (синхронно-реактивные модели),
- *rendezvous*-ориентированные модели,
- 3-D визуализация,
- *continuous-time* (модели непрерывного времени).

И другие, в том числе экспериментальные.







Эта модель содержит три актора, каждый из которых имеет один порт. Актор A посылает сообщения актору B и C через свой порт. Ромб Relation указывает, что выходное сообщение от A доставляется и в B, и в C.

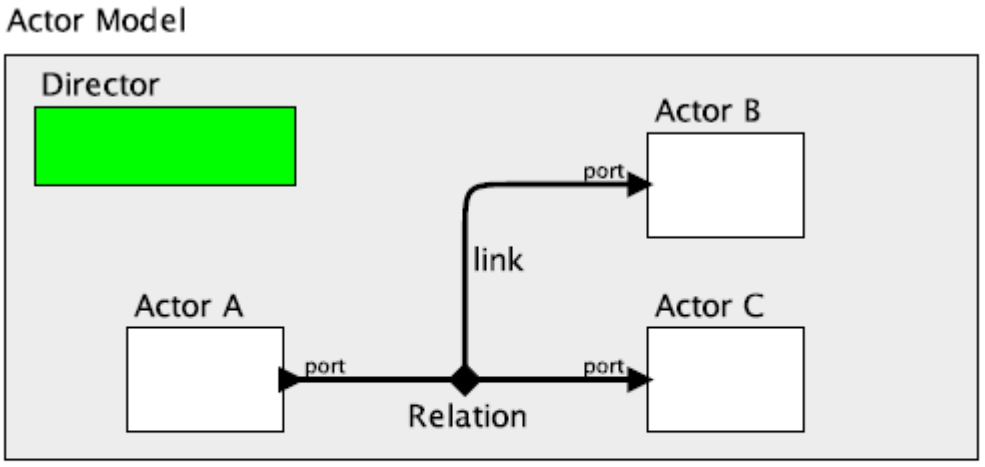


Figure 1.3: Visual rendition of a simple actor model.

- *«Каждый уровень иерархии в модели может иметь своего собственного директора, и отдельные директора могут быть составлены иерархически.»*
- *На верхнем уровне иерархии обязательно должен быть директор, т. к. он управляет структурной схемой.*
- *На следующих уровнях иерархии актор может быть прозрачным (т. е. управляться директором с более высокого уровня иерархии).*

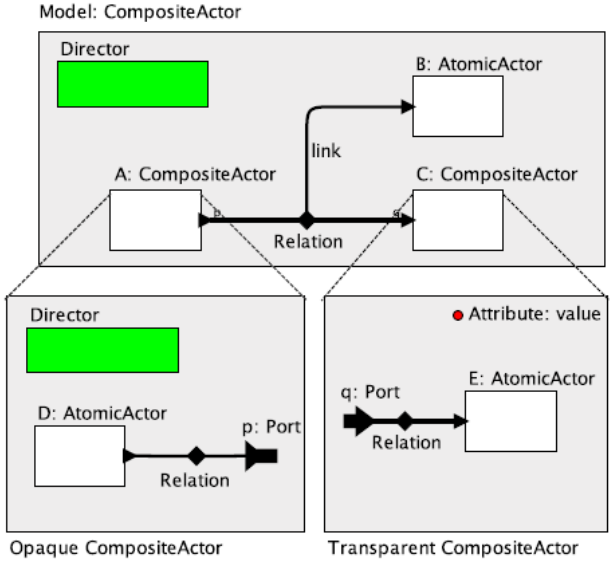


Figure 1.4: A hierarchical actor model consisting of a top-level composite actor and two submodels, each of which is also a composite actor.

Знакомство с Ptolemy II (2)

«Основной акцент в проекте был сделан на понимании разнородных комбинаций моделей вычислений, реализованных директорами. Директора могут быть иерархически объединены с конечными автоматами для создания **«режимных» моделей (modal models)**. Иерархическая комбинация моделей непрерывного времени с конечными автоматами дает гибридные системы; комбинация синхронно-реактивных моделей с конечными автоматами дает StateCharts (вариант Ptolemy II близок к SyncCharts).»

«Режимные» модели (modal models)

- **«Режимная» модель** (Modal Model) — это явное представление конечного набора поведений (режимов) и правил, которые управляют переходами между ними. Правила фиксируются конечными автоматами.
- Для реализации «режимных» моделей используется актор ModalModel.
- **Актор ModalModel** — это иерархический актор, подобный составному актору, но с несколькими определениями моделей (уточнениями, refinement) вместо одного. Каждое уточнение определяет один режим поведения, и конечный автомат определяет, какое уточнение активно в любой момент времени.

Если не использовать уточнения в состояниях, то ModalModel вырождается в **конечный автомат (FSM)** — набор состояний и правил, которые управляют переходами между ними:

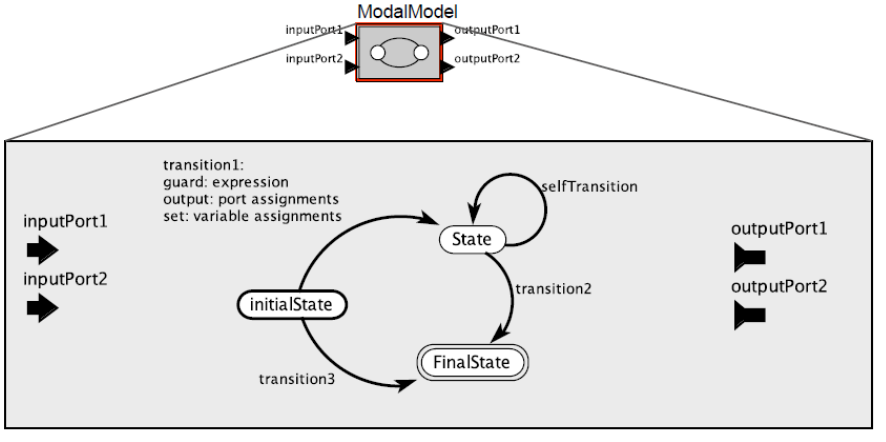
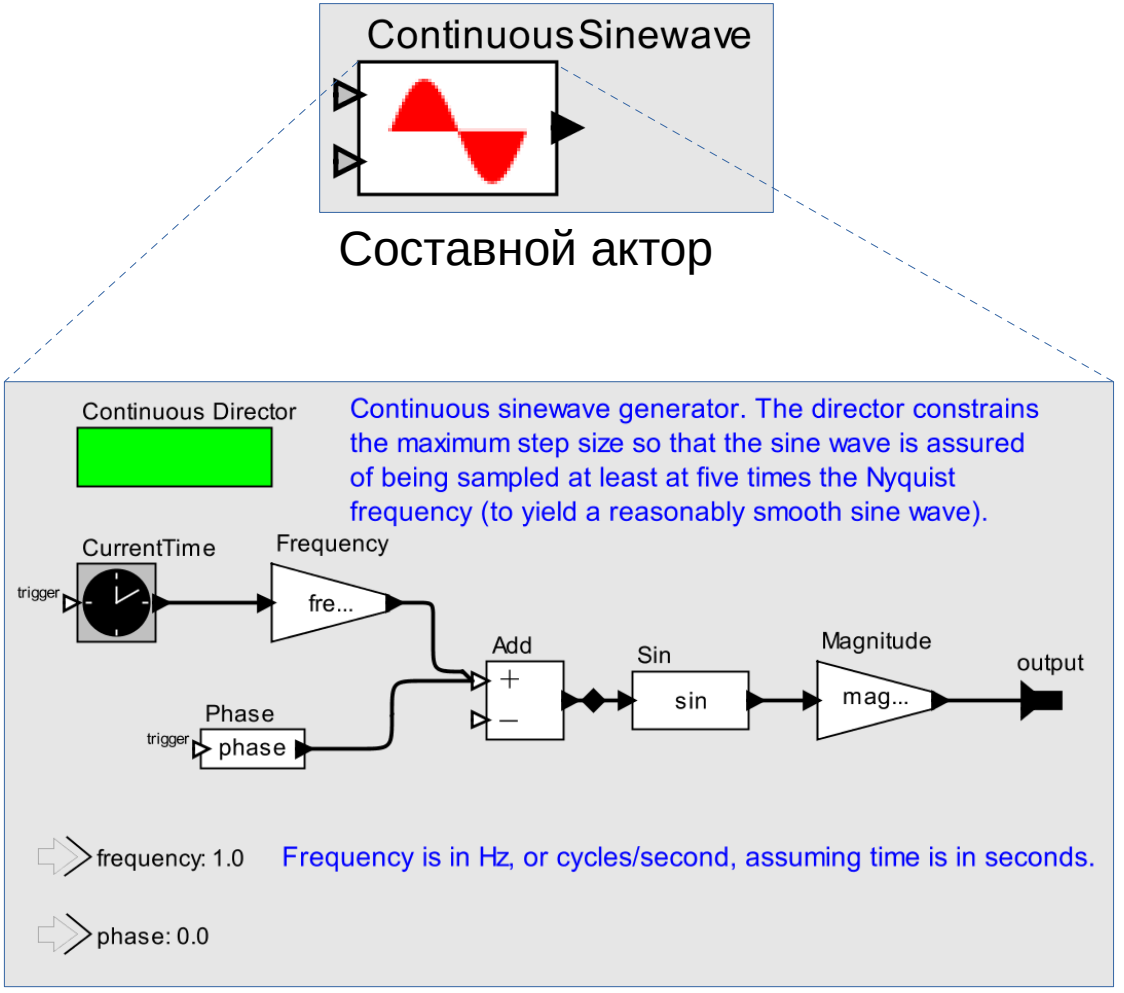


Figure 6.1: Visual notation for state machines in Ptolemy II.



Уточнение (определение модели) составного актора ContinuousSinewave

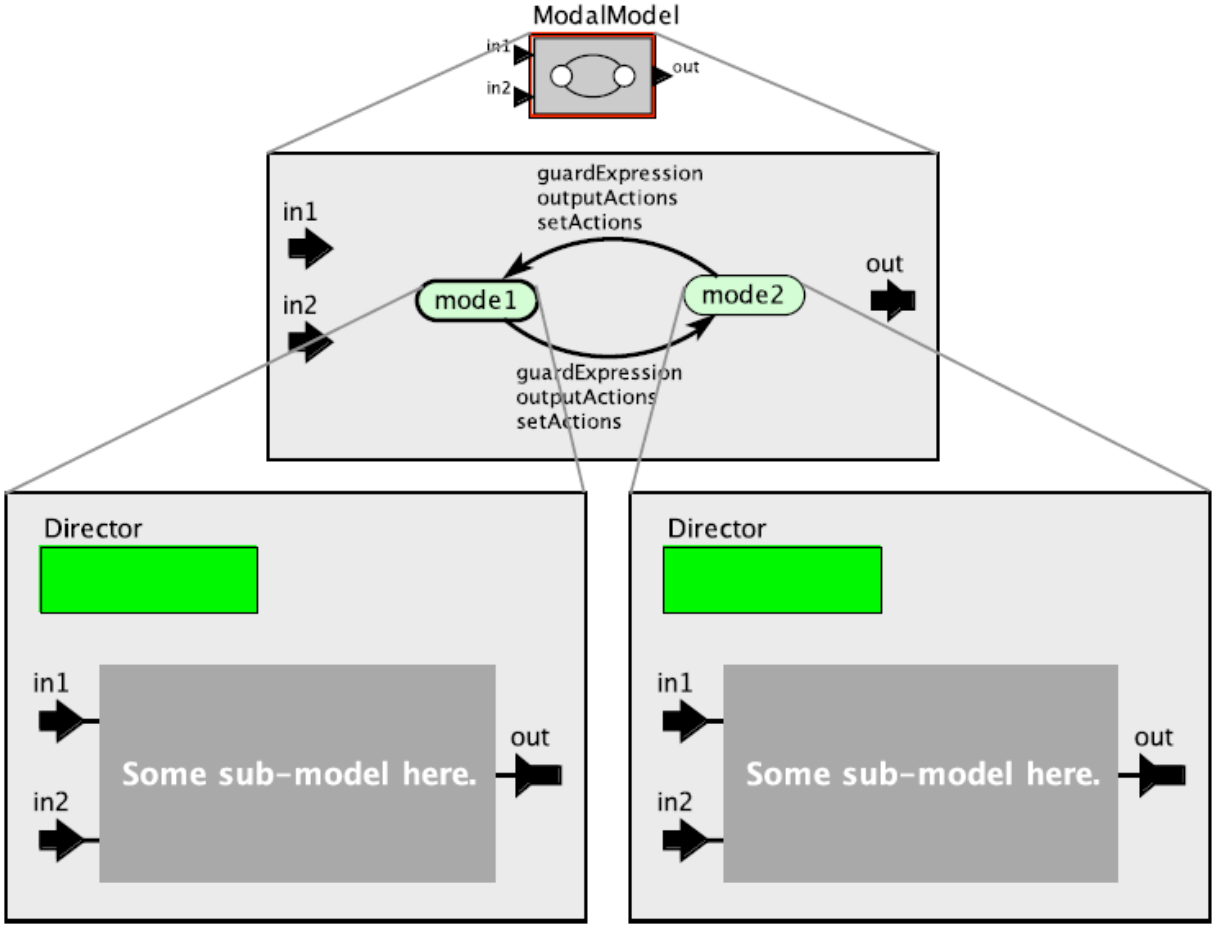
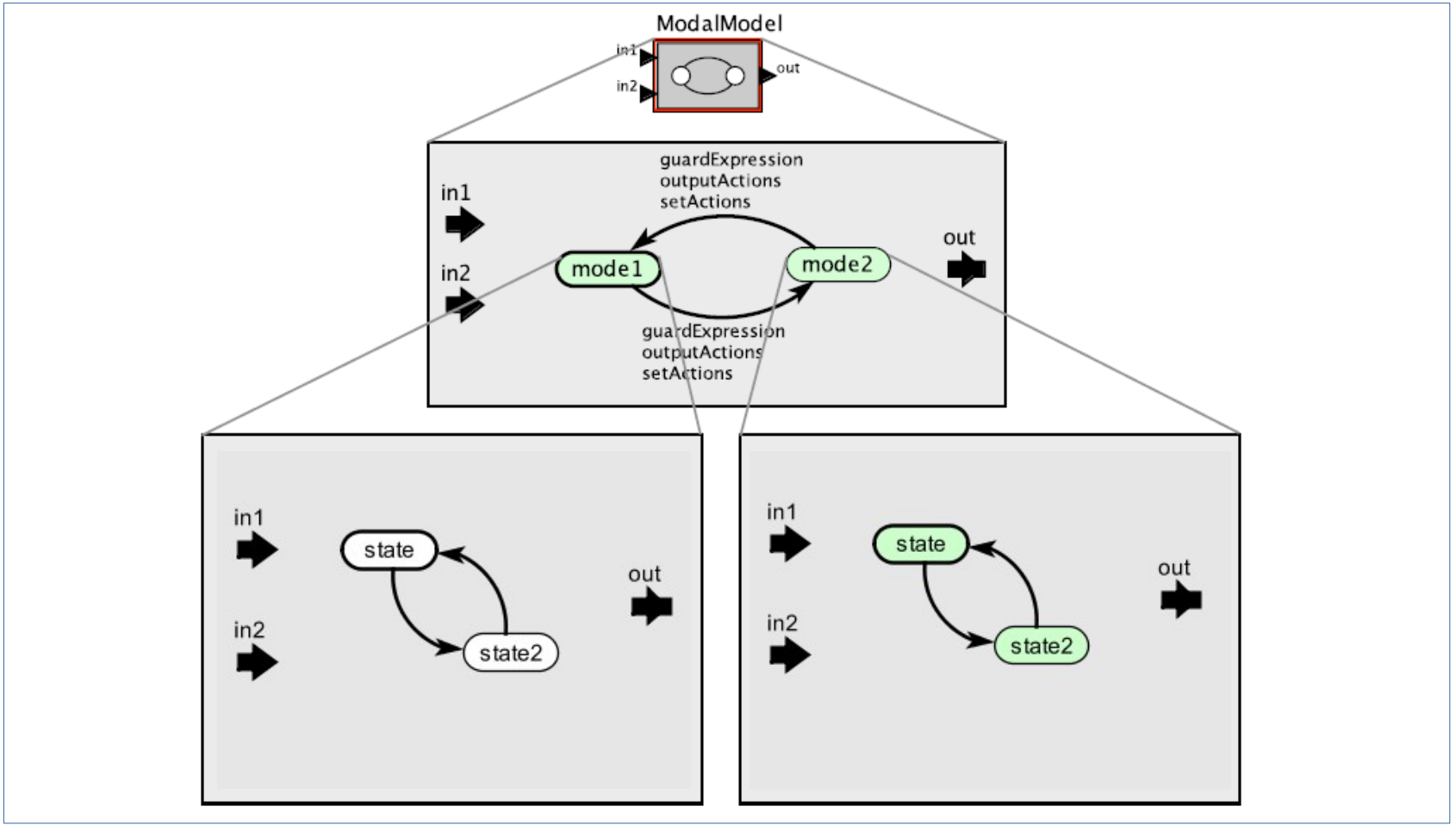
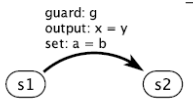
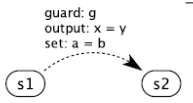
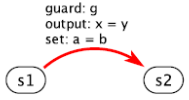
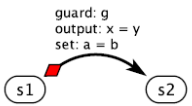
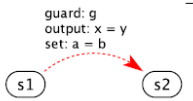
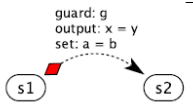


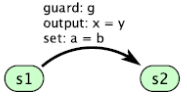
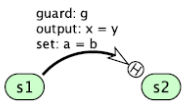
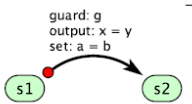
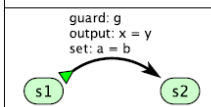
Figure 8.4: General pattern of a modal model with two modes, each with its own refinement.


ModalModel
in1
in2
out
guardExpression
outputActions
setActions
mode1
mode2
guardExpression
outputActions
setActions
state
state2
state
state2

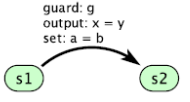
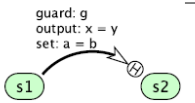
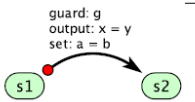
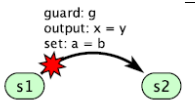
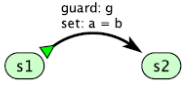
В зависимости от типа состояния возможны различные типы переходов КА, нестандартные для UML

notation	description
	An **ordinary transition**. Upon firing, if the guard g is `true` (or if no guard is specified), then the FSM will choose the transition and produce the value y on output x. Upon transitioning, the actor will set the variable a to have value b.
	A **default transition**. Upon firing, if no other non-default transition is enabled and the guard g is `true`, then the FSM actor will choose this transition, produce outputs, and set variables in the same manner as above.
	A **nondeterministic transition** This transition allows another nondeterministic transition to be enabled in the same iteration. One of the enabled transitions will be chosen nondeterministically.
	An **immediate transition**. If state *s1* is the current state, then this is like an ordinary transition. However, if state *s1* is the destination state of some transition that will be taken and the guard g is `true`, then the FSM will also immediately transition to *s2*. In this case, there will be two transitions in a single iteration. The output x will be set to value y upon firing, and the variable a will be set to b upon transitioning. If more than one transition in a chain of immediate transitions sets an output or variable, then the last transition will prevail.
	A **nondeterministic default transition**. A nondeterministic transition with the (lower) priority of a default transition.
	An **immediate default transition**. An immediate transition with the (lower) priority of a default transition, compared with other immediate transitions.

Сводка переходов FSM и их обозначений, которые можно комбинировать для обозначения комбинаций типов переходов. Например, недетерминированный немедленный переход по умолчанию будет окрашен в красный цвет, иметь начальный ромб и изображаться пунктирными линиями.

notation	description
	An **ordinary transition**. Upon firing, the refinement of the source state is fired first, and then if the guard g is `true` (or if no guard is specified), then the FSM will choose the transition. It will produce the value y on output port x, overwriting any value that the source state refinement might have produced on the same port. Upon transitioning (in postfire), the actor will set the variable a to have value b, again overwriting any value that the refinement may have assigned to a. Finally, the refinements of state *s2* are reset to their initial states. For this reason, these transitions are sometimes called **reset transitions**.
	A **history transition**. This is similar to an ordinary transition, except that when entering state *s2*, the refinements of that state are *not* reset to their initial states, but rather resume from whatever state they were in when the refinement was last active. On first entry to *s2*, of course, the refinements will start from their initial states.
	A **preemptive transition**. If the current state is *s1* and the guard is true, then the state refinement (the FSM sub-model) for *s1* will not be invoked prior to the transition.
	A **termination transition**. If all refinements of state *s1* reach a final state and the guard is true, then the transition is taken.

Сводка переходов FSM и их обозначений для иерархических машин состояний. Здесь предполагается, что все уточнения сами являются FSM.

notation	description
	An **ordinary transition**. Upon firing, the refinement of the source state is fired first, and then if the guard g is `true` (or if no guard is specified), then the FSM will choose the transition. It will produce the value y on output port x, overwriting any value that the source state refinement might have produced on the same port. Upon transitioning (in postfire), the actor will set the variable a to have value b, again overwriting any value that the refinement may have assigned to a. Finally, the refinements of state *s2* are initialized. For this reason, these transitions are sometimes called **reset transitions**.
	A **history transition**. This is similar to an ordinary transition, except that when entering state *s2*, the refinements of that state are *not* initialized. On first entry to *s2*, of course, the refinements will have been initialized.
	A **preemptive transition**. If the current state is *s1* and the guard is true, then the state refinement for *s1* will not be iterated prior to the transition.
	An **error transition**. If any refinement of state *s1* throws an exception or a model error, and the guard is true, then this transition will be taken. The output and set actions of the transition can refer to special variables *errorMessage*, *errorClass*, and *errorCause*, as explained in Section 8.2.3.
	A **termination transition**. If all refinements of state *s1* have returned false on postfire, and the guard is true, then the transition is taken. Notice that since it cannot be known until the postfire phase that this transition will be taken, the transition cannot produce outputs. For most domains, postfire is too late to produce outputs. Moreover, this transition has lower priority than all other transitions, including default transitions, because it cannot become enabled until postfire.

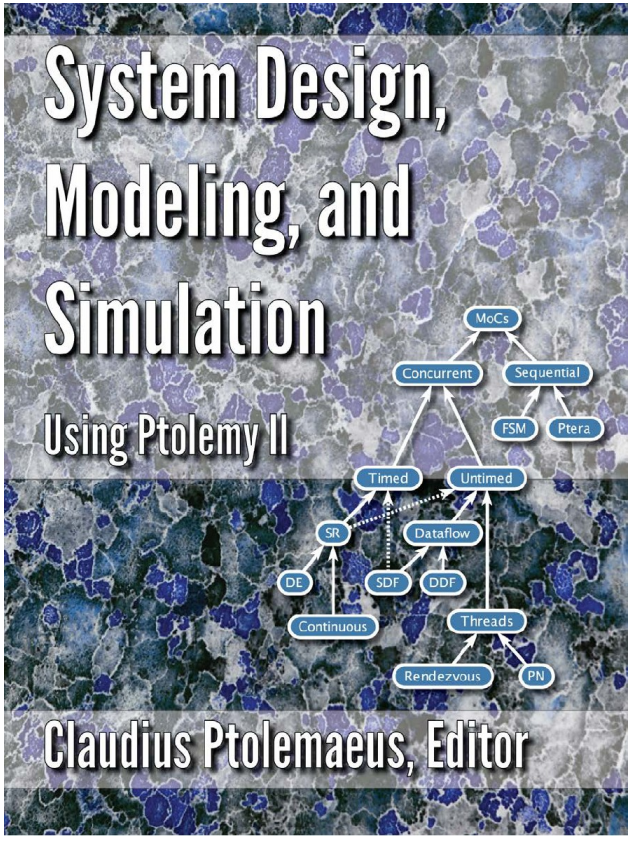
Краткое описание переходов «режимной» модели и их обозначений. Здесь предполагается, что уточнения состояний - это произвольные модели Ptolemy II, каждая из которых имеет директора.

Знакомство с Ptolemy II (3)

- «Основой Ptolemy II является набор Java-классов и пакетов, многоуровневых для предоставления все более специфичных возможностей. Пакет kernel поддерживает абстрактный синтаксис, иерархическую структуру объектов с портами и взаимосвязями. Графический редактор под названием Vergil поддерживает визуальное редактирование этого абстрактного синтаксиса. Конкретный синтаксис XML, называемый MoML, обеспечивает постоянный формат файла для моделей.»
- «На основе этого фреймворка были созданы различные специализированные инструменты:
  - HyVisual (для моделирования гибридных систем),
  - Kepler (для научных рабочих процессов),
  - VisualSense (для моделирования беспроводных сетей),
  - Viptos (для проектирования сенсорных сетей),
  - CapeCode (для проектирования приложений Интернета Вещей)
  - и некоторые коммерческие продукты.»
- «Ключевые части инфраструктуры включают абстрактную семантику актёра, которая обеспечивает взаимодействие различных моделей вычислений с четко определенной семантикой; модель времени (в частности, сверхплотное время, которое обеспечивает взаимодействие непрерывной динамики и императивной логики); и сложная система типов, поддерживающая проверку типов, вывод типов и полиморфизм.»

Ptolemy II. Учебное пособие

- Claudius Ptolemaeus, Editor, ***System Design, Modeling, and Simulation Using Ptolemy II***, Ptolemy.org, 2014.
- https://ptolemy.berkeley.edu/books/Systems/
- В представленной книге Ptolemy II используется в качестве основы для обсуждения моделирования. Рассматриваются различные модели вычислений, приводятся практические примеры с использованием Ptolemy II через его графический интерфейс Vergil.
- Подробнее об упомянутых, но нераскрытых аспектах можно узнать в следующих главах:
  - Различным моделям вычислений, имеющие хорошо формализованную семантику в Ptolemy II посвящена вся вторая глава.
  - «Режимные» модели, её возможные комбинации с различными моделями вычслений — глава 2, Modal Models.
  - Иерархические конечные автоматы — глава 2, FSM, Hierarchical FSMs.
  - Абстрактные синтаксис и семантика, абстрактная семантика актора — глава 3, Software Architecture.
  - Язык выражений — глава 3, Expressions.
  - Система типов — глава 3, The Type System.



Резюме

- **Фреймворк** = среда моделирования
- **Актор** = активный объект UML с ограничениями на тип взаимодействия между объектами
- **Акторно-ориентированный дизайн -** подход к моделированию, в котором акцент ставится на моделировании систем как набора независимых компонентов, которые взаимодействуют друг с другом путём обмена сообщениями. *Взаимодействующие через порты активные объекты.*
- **Модель вычислений** — набор правил, которые управляют одновременным выполнением компонентов и обменом данными между компонентами.
- **Директор** — управляющий компонент, реализация конкретной модели вычислений.

Среды и технологии

- При рассмотрении различных сред моделирования в процессе обучения возникают такие понятия, как подходы к моделированию, например, *модельно-ориентированный подход* или *объектно-ориентированный подход*.

Существующие подходы к моделированию

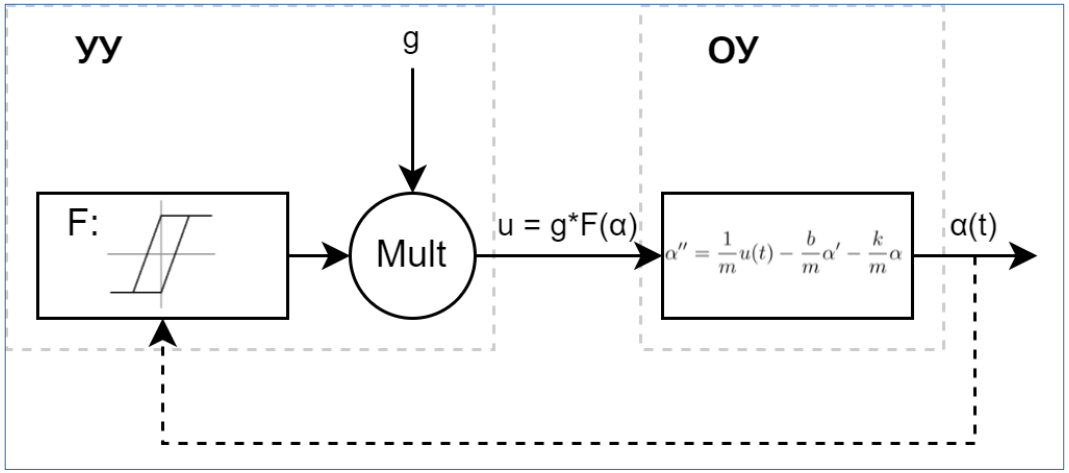
- **Модельно-ориентированный (МО)** подход концентрируется на процессе разработки (создание и симуляция) и интеграции (генерация исполняемого кода) модели. Данный подход более выражен в коммерческих средах, например, в Simulink.
- **Объектно-ориентированный (ОО)** подход задействует принципы создания и наследования классов, инкапсуляции, полиморфизма с целью повторного использования компонентов в новых компонентах и упрощении коллективного проектирования. Представитель с явно выраженным ОО подходом — AnyDynamics.
- В обсуждаемом в данной презентации инструменте Ptolemy II основным подходом является **акторно-ориентированный (АО)** подход. В данном подходе акцент ставится на моделировании систем как набора независимых компонентов, которые взаимодействуют друг с другом путём обмена сообщениями.

Существующие подходы к моделированию

- Аспекты различных подходов реализуются в той или иной мере в средах моделирования в зависимости от целей разработчиков.
- Рассмотрим эти аспекты в Simulink, AnyDynamics и Ptolemy II.

Инструмент	МО аспект	ОО аспект	АО аспект
Simulink	*Основной подход.* Создание, симуляция и верификация моделей. Инструменты отладки модели. Генерация кода C, C++, CUDA, PLC, Verilog и VHDL	Представлен в неявном виде: создание новых компонентов как композиции типовых компонентов (экземпляров классов) из обширной библиотеки.	Имеет синтаксически идентичное представление компонентов с Ptolemy II (компоненты с входами-выходами), но фиксированный набор моделей взаимодействия.
AnyDynamics	Создание, симуляция и верификация моделей. Инструменты отладки модели. Нет генерации кода, но есть возможность создания встраиваемой модели (динамическая библиотека DLL) в коммерческой версии среды моделирования (Rand Model Designer Professional).	*Основной подход.* Поддерживается в полной мере, позволяя создавать и наследовать классы, добавлять новые компоненты, переопределять все унаследованные атрибуты, использовать инкапсуляцию, полиморфизм.	Компоненты параллельны и взаимодействуют через порты, но порты не являются ни входами, ни выходами. Вместо этого связи между портами являются ограничениями на переменные. Имеет фиксированный набор моделей взаимодействия.
Ptolemy II	Создание, симуляция и верификация моделей. Инструменты отладки модели. Экспериментальная возможность генерации C и Java кода.	Представлен в упрощённом виде: поддерживается наследование классов, но в классе-наследнике возможно только добавлять новые компоненты и переопределять только параметры.	*Основной подход.* Акторы — это компоненты, которые выполняются одновременно и взаимодействуют посредством сообщений, отправляемых через взаимосвязанные порты. Модель их взаимодействия не является предопределённой, а определяется специальным компонентом.

Пример: Модель затухающего гармонического осциллятора с воздействием внешней силы

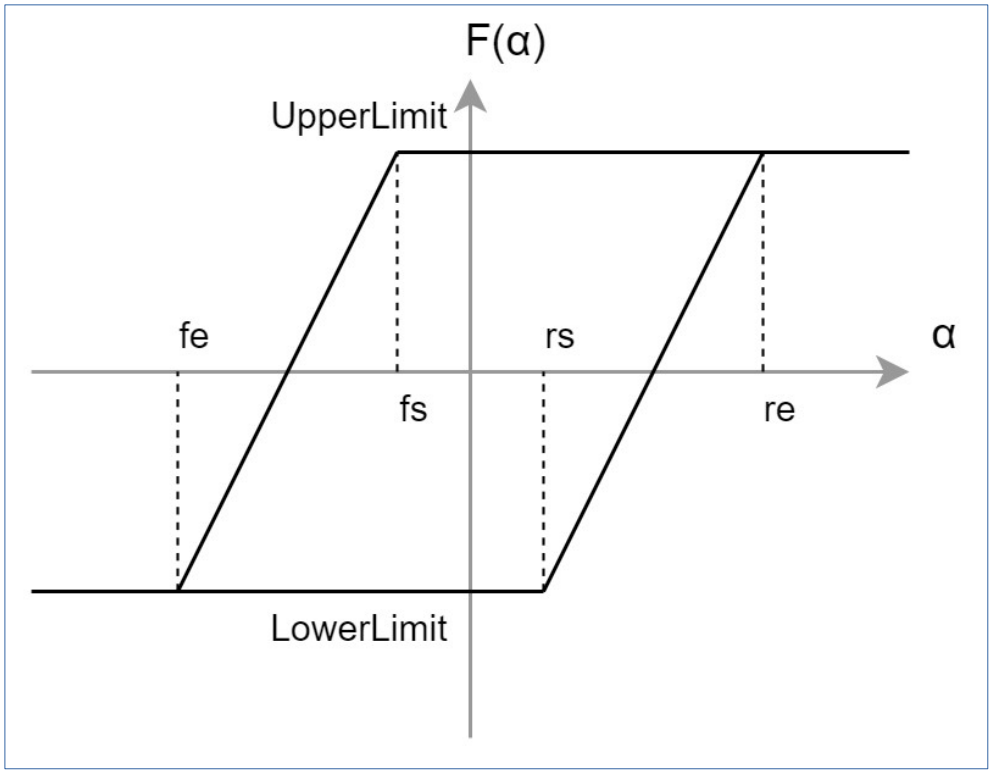


$$\alpha'' = \frac{g}{m}u(t) - \frac{b}{m}\alpha' - \frac{k}{m}\alpha$$

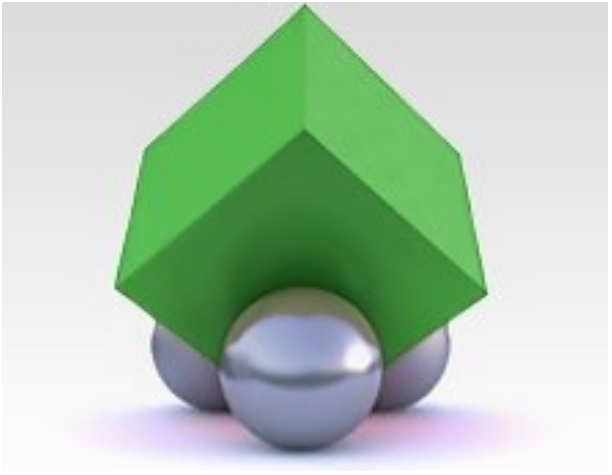
где $u(t)$ - внешняя сила,
$b\alpha'$ - слагаемое затухания,
$k\alpha$ - сила упругости нити,
m - масса грузика,
g - коэффициент усиления внешней силы.

Нелинейность «насыщение с гистерезисом» с параметрами:

- верхний предел (UpperLimit),
- нижний предел (LowerLimit),
- начало отрезка возрастания (rs, rising_start),
- конец отрезка возрастания (re, rising_end),
- начало отрезка убывания (fs = -rs, falling_start)
- и конец отрезка убывания (fe = -re, falling_end).



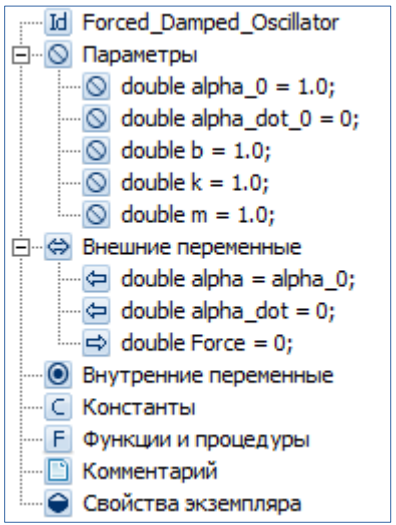
Создадим компоненты
«Затухающих осциллятор с воздействием внешней силы»
и нелинейную функцию «Насыщение с гистерезисом»
в средах AnyDynamics и Ptolemy II.



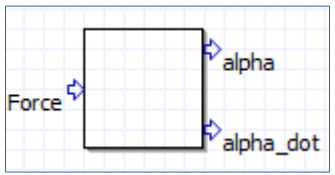
Создание компонента «Затухающий гармонический осциллятор с воздействием внешней силы»

[AnyDynamics]

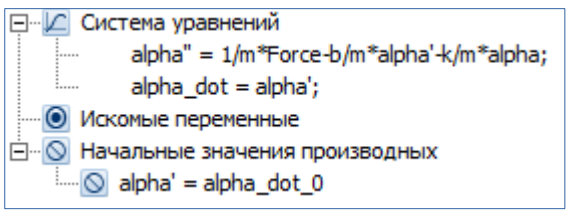
Создаём класс со следующим стереотипом: Непрерывный, Элементарный объект, Открытая система и редактируем следующие элементы:



Элементы описания класса



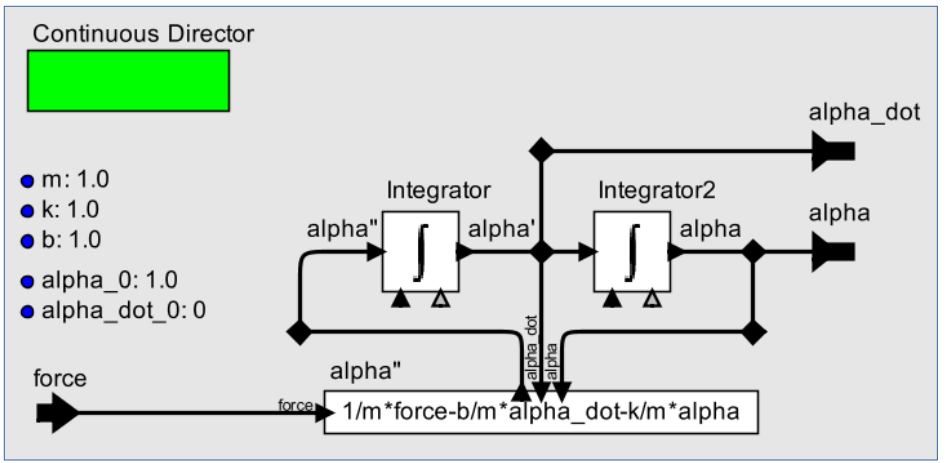
Окно «Интерфейс»



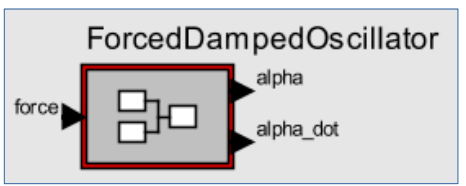
Окно «Уравнения»

[Ptolemy II]

Из библиотеки компонентов переносим в рабочую область актор CompositeActor и создаём следующую модель внутри этого компонента:



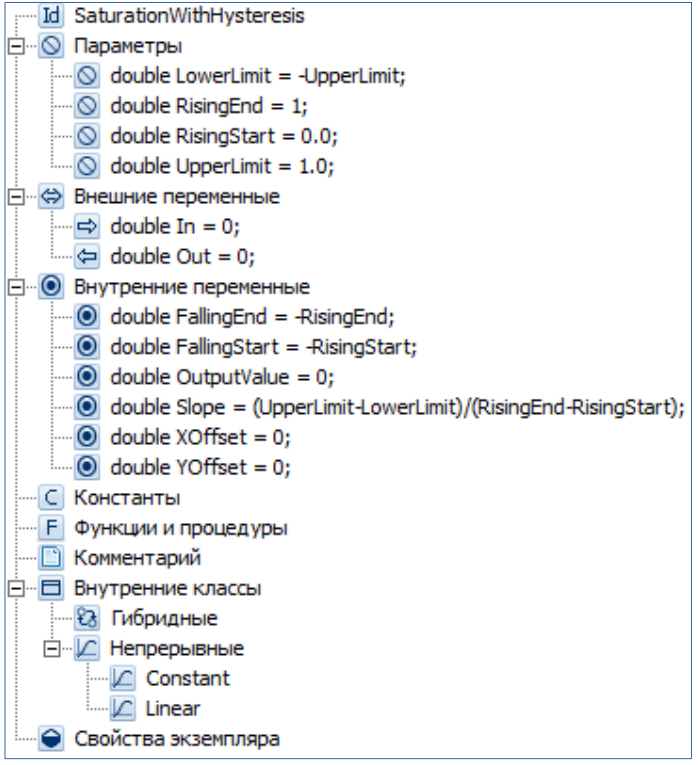
Уточнение созданного CompositeActor



Создание компонента Нелинейная функция «Насыщение с гистерезисом»

[AnyDynamics]

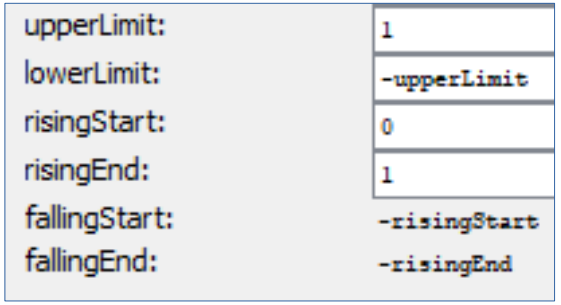
Создаём класс со следующим стереотипом: Гибридный, Элементарный объект, Открытая система и редактируем следующие элементы:



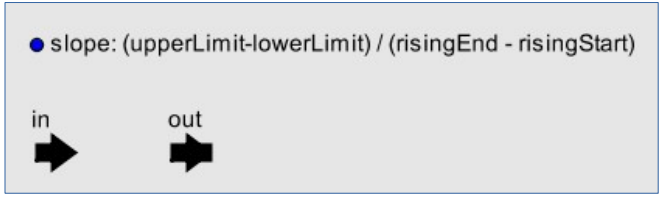
Элементы описания класса

[Ptolemy II]

Из библиотеки компонентов переносим в рабочую область актор ModalModel и через контекстное меню (Customize->Configure->Add) создаём параметры:



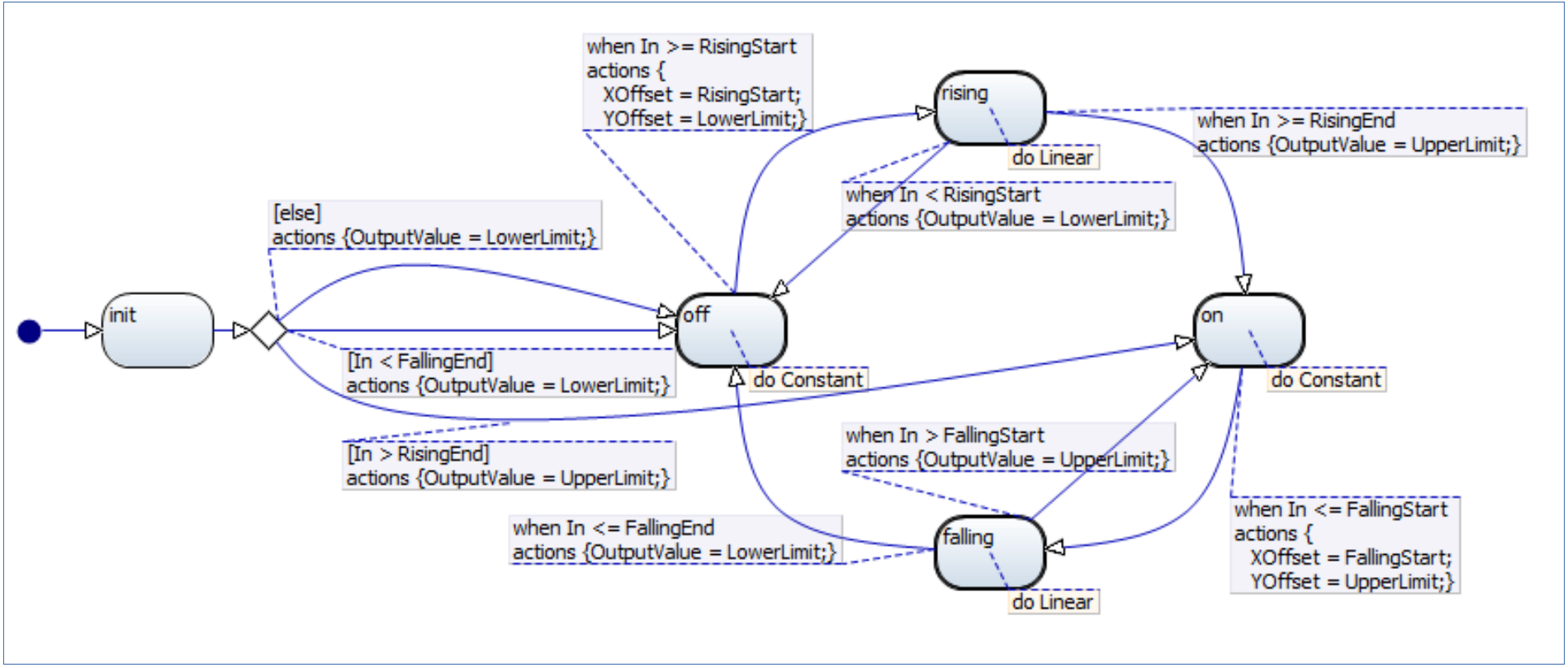
Внутри этого актора добавляем входной и выходной порты и параметр slope:



Создание компонента нелинейной функции «Насыщение с гистерезисом». Гибридный автомат в AnyDynamics (1)

В окне «Карта поведения» создаём гибридный автомат, описывающий нелинейную функцию.

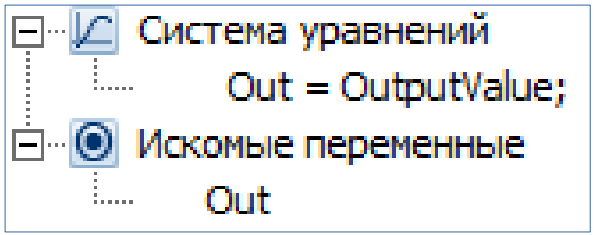
Переход на начальный «отрезок» функции происходит с помощью блока «Точка ветвления» и условий для перехода на нижний (off) или верхний (on) отрезки функции. В случае, если начальное значение входного порта находится между точками конца отрезков rising и falling, то переход осуществляется по ветке *else* на отрезок off.



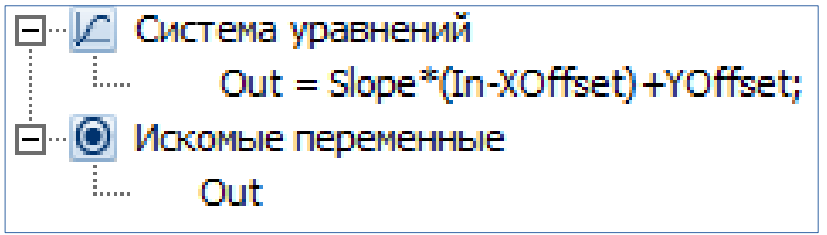
Создание компонента нелинейной функции «Насыщение с гистерезисом». Гибридный автомат в AnyDynamics (2)

Создадим непрерывные деятельности на отрезках on, off, rising и falling нелинейной функции.

1) В описании элементов класса в разделе Внутренние классы → Непрерывные создадим класс Constant, в котором в окне «Уравнения» введём уравнение `Out = OutputValue`



2) В описании элементов класса в разделе Внутренние классы → Непрерывные создадим класс Linear, в котором в окне «Уравнения» введём уравнение `Out = Slope*(In-XOffset)+YOffset`

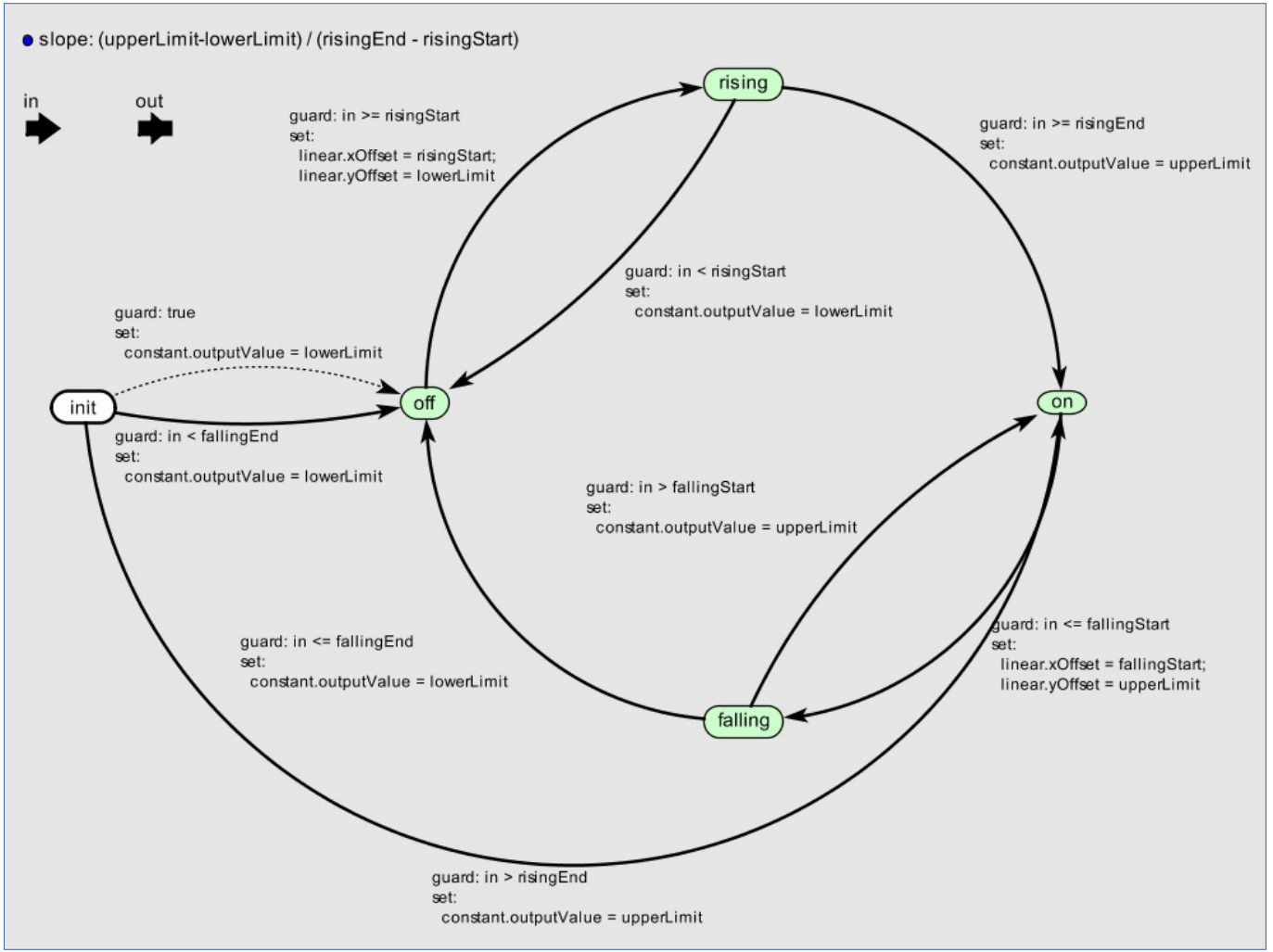


3) Добавим класс Constant как деятельность в состояниях on и off, а Linear как деятельность в состояниях rising и falling.

Создание компонента нелинейной функции «Насыщение с гистерезисом». Гибридный автомат в Ptolemy II (1)

В акторе ModalModel создаём гибридный автомат, описывающий нелинейную функцию.

Здесь переход на начальный «отрезок» функции происходит с помощью условий для перехода на нижний (off) или верхний (on) отрезки функции. В случае, если начальное значение входного порта находится между точками конца отрезков rising и falling, то переход осуществляется по переходу по умолчанию (переход с пунктирной линией, имеющий приоритет ниже обычного перехода со сплошной линией) на отрезок off.



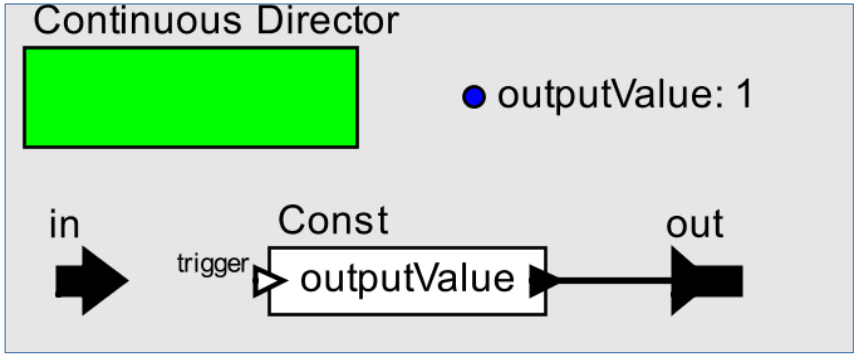
Создание компонента нелинейной функции «Насыщение с гистерезисом». Гибридный автомат в Ptolemy II (1)

Создадим непрерывные деятельности на отрезках on, off, rising и falling нелинейной функции.

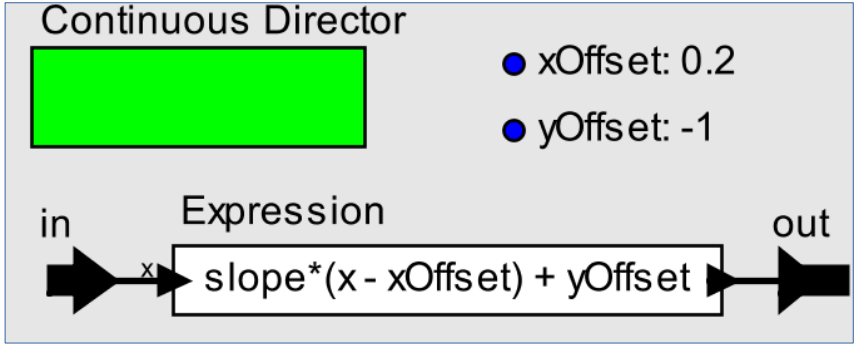
Чтобы *создать* деятельность (уточнение, refinement в терминах Ptolemy II) необходимо на одном из состояний вызвать контекстное меню и выбрать Add Refinement. Чтобы *добавить* эту же деятельность другим состояниям необходимо вызвать контекстное меню (Customize->Configure) у нужного состояния и задать имя уже созданной деятельности в параметре refinementName.

1) В состоянии off создадим класс Constant и также добавим его в состояние on.

2) В состоянии rising создадим класс Linear и также добавим его в состояние falling.



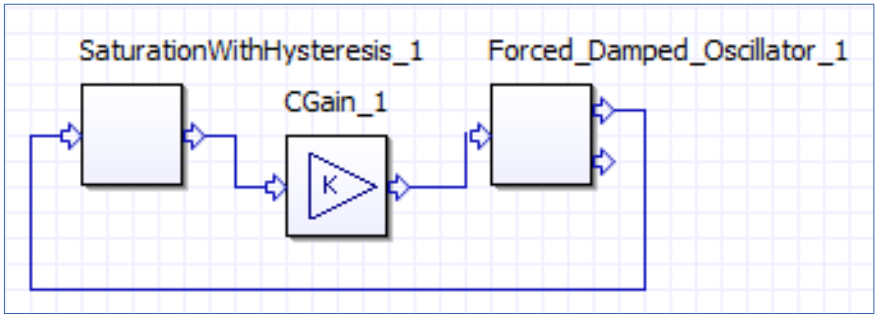
Уточнение Constant

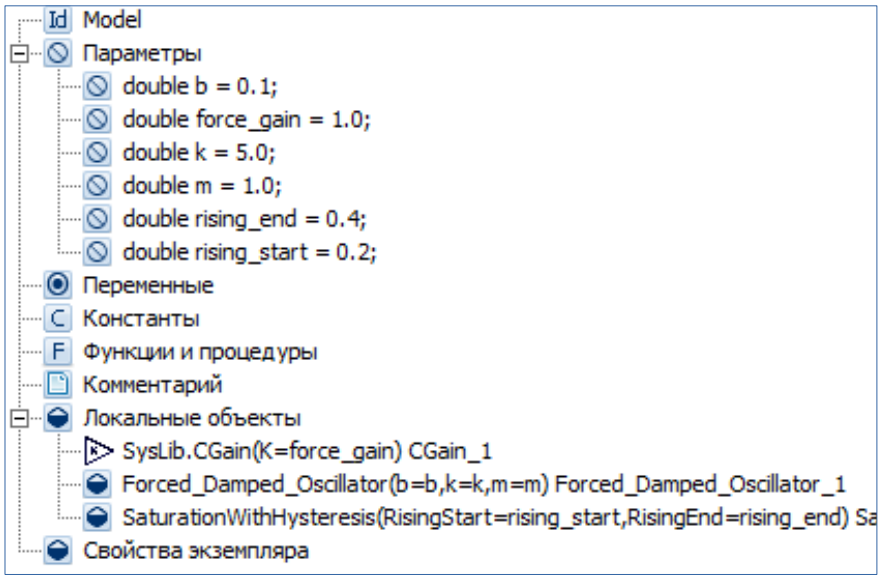


Уточнение Linear

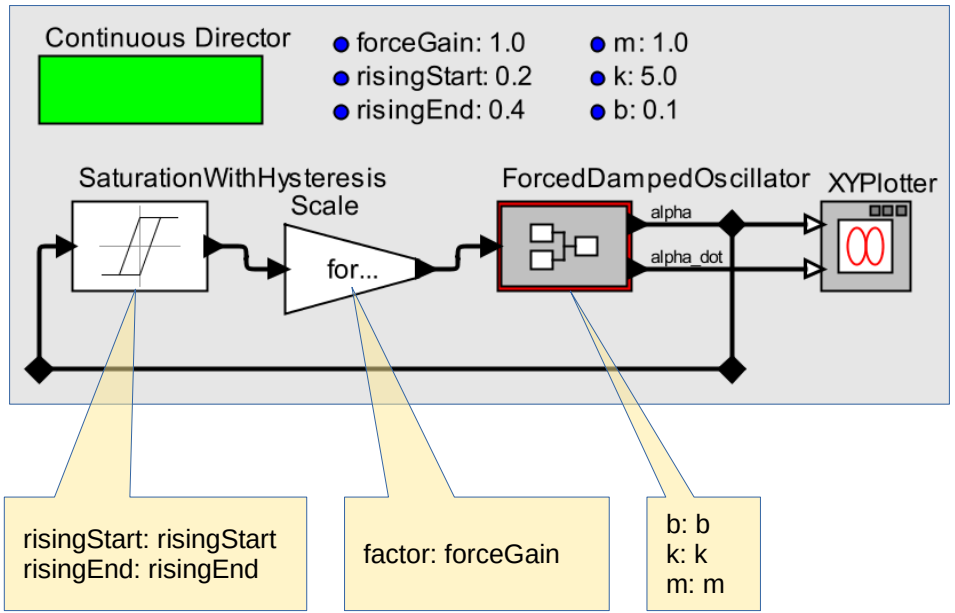
Создадим модели типа «Компоненты с входами-выходами» из созданных компонентов (и класса «Усилитель» (CGain и Scale) из стандартных библиотек сред). Зададим параметры b, k и m осцилляторов, risingStart и risingEnd нелинейной функции и множитель компонента «Усилитель».

[AnyDynamics]





[Ptolemy II]



Установим параметры выполнения моделей и сравним фазовые портреты с разными режимами работы (с разными коэффициентами в системе уравнений).

Установка параметров выполнения моделей

[AnyDynamics]

[Ptolemy II]

Решатель ОДУ

Визуальная модель → Установки → Численные методы... → Дифференциальные уравнения

Установить RK45

В контекстном меню Continuous Director:
Customize → Configure → ODESolver

Установить ExplicitRK45Solver

Погрешность

Визуальная модель → Установки → Численные методы… → Погрешности

Установить абсолютную погрешность для переменной 1.000E-06

В контекстном меню Continuous Director:
Customize → Configure

Установить errorTolerance 1e-6

Шаг интегрирования

Визуальная модель → Установки → Численные методы… → Погрешности

Установить Фиксированный шаг 1.000E-02

В контекстном меню Continuous Director:
Customize → Configure

Установить initStepSize и maxStepSize 1e-2

Модельное время

Визуальная модель → Сервис → Условия останова → По времени

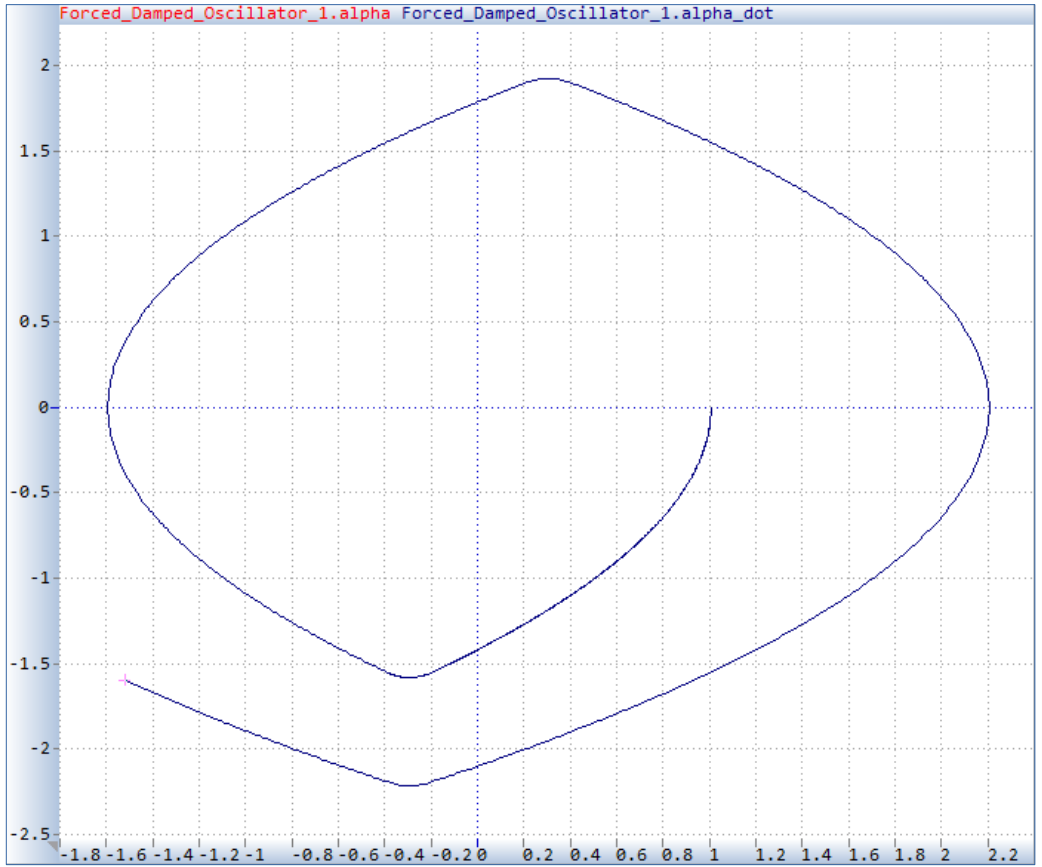
Установить Останов на 10

В контекстном меню Continuous Director:
Customize → Configure → stopTime

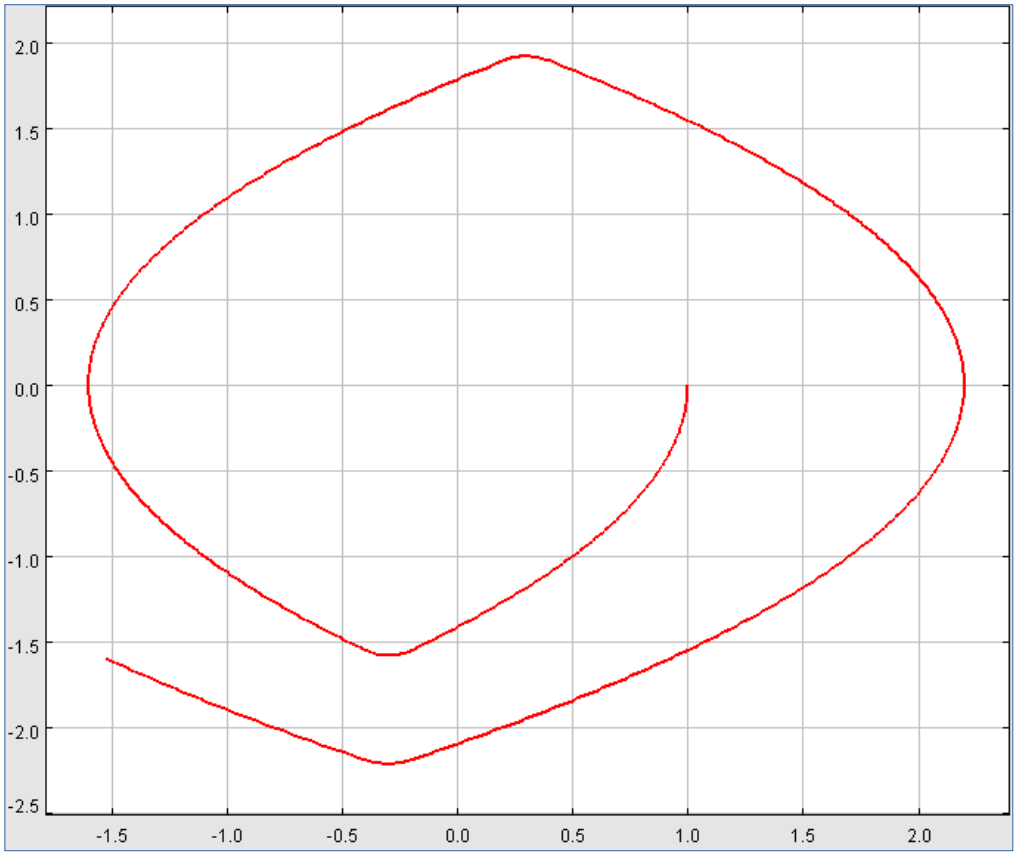
Установить 10

force_gain = -1.0, b = 0.0, k = 0.0
(только нелинейная функция)

[AnyDynamics]

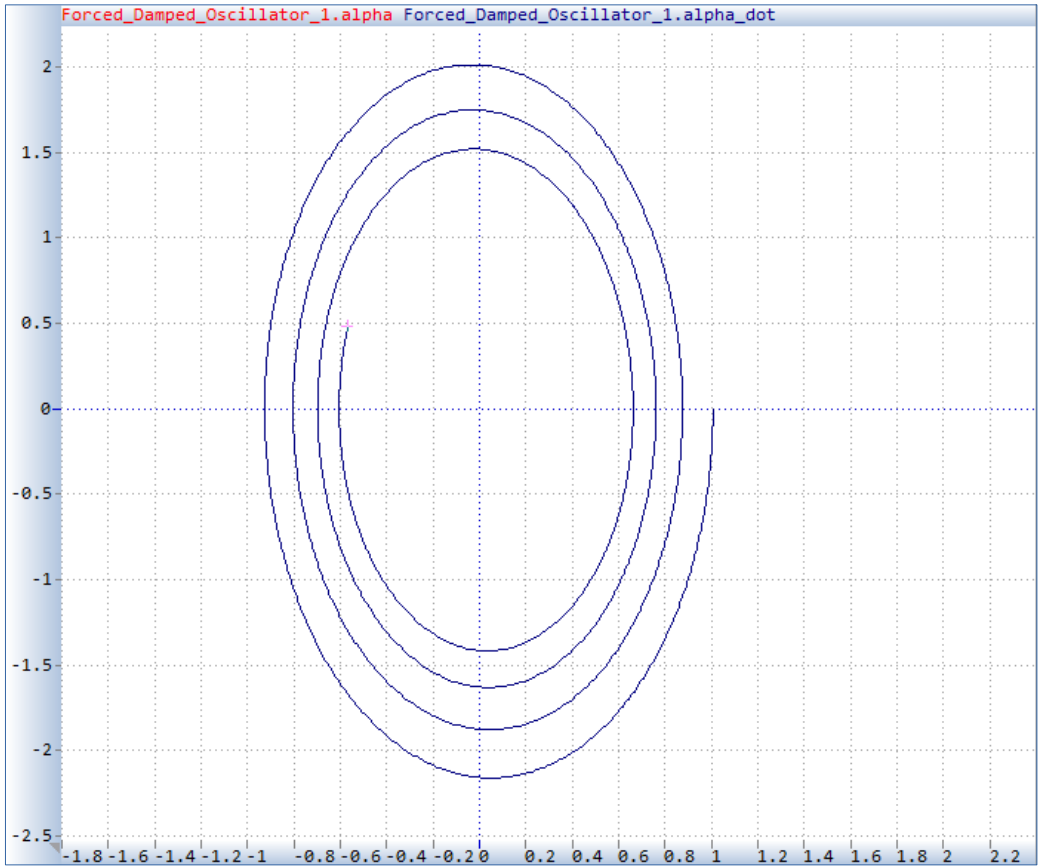


[Ptolemy II]

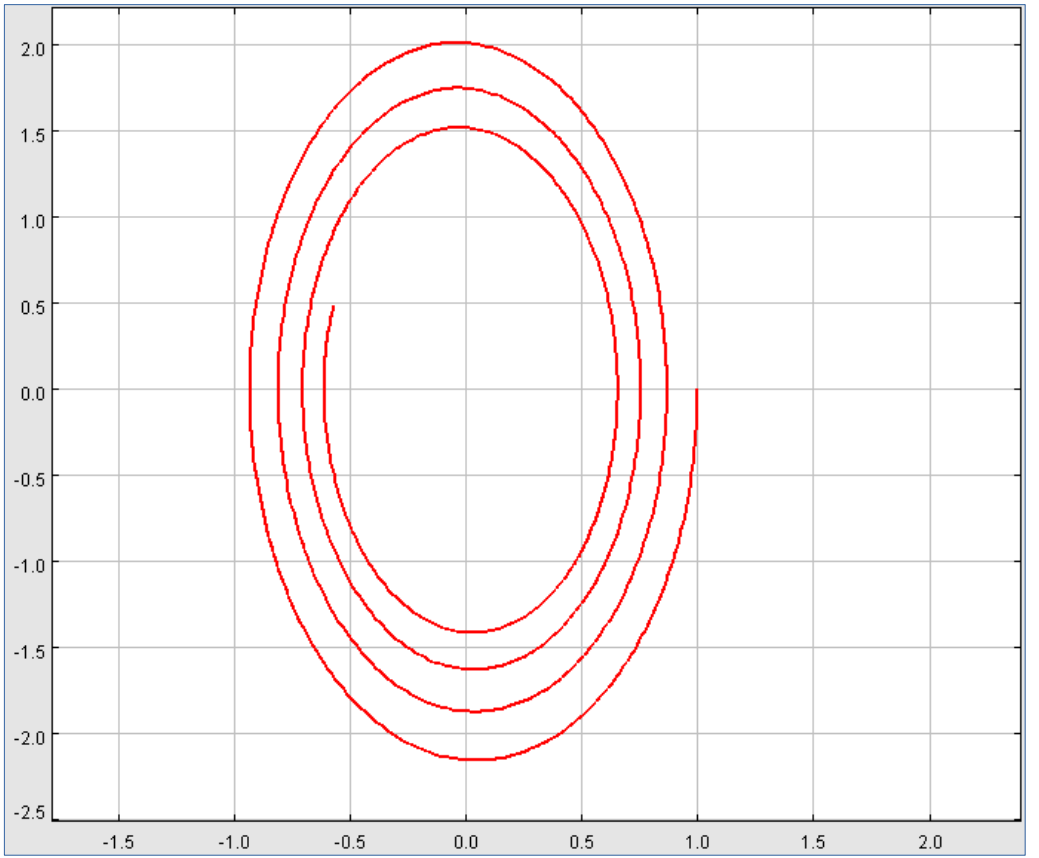


force_gain = 0.0, b = 0.1, k = 5.0
(только затухающий осциллятор)

[AnyDynamics]



[Ptolemy II]



force_gain = -1.0, b = 0.1, k = 5.0
(затухающий осциллятор с воздействием внешней силы)