

ANYLOGIC

Справочное руководство по библиотеке SMO

Оглавление

Общие принципы работы библиотекой	3
Сборка блок-схемы из объектов.....	3
Правила пересылки заявок.....	5
Динамические параметры	7
Базовый класс Entity	7
Создание подклассов класса Entity	10
Source.....	10
Sink	13
Enter	14
Exit	15
SelectOutput.....	16
Queue	17
Delay	20
Resource.....	23
SeizeQ	25
Release	27
ProcessQ.....	29
ProcessQ_b.....	31
Gulat.....	31
Process	32
Process_b	33
Func.....	33
Prerivanie	34

Общие принципы работы библиотекой

AnyLogic™ — уникальный инструмент имитационного моделирования, поддерживающий на единой платформе абсолютно все существующие подходы дискретно-событийного и непрерывного моделирования. AnyLogic™ имеет исключительно развитый базовый язык дискретного и смешанного дискретно-непрерывного моделирования, на основе которого и построена библиотека Enterprise Library.

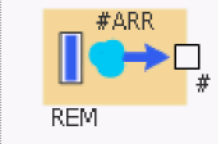
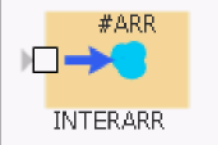

Библиотека AnyLogic™ Enterprise Library предоставляет высокоуровневый интерфейс для быстрого создания дискретно-событийных моделей с помощью блок-схем. Графическое представление систем с помощью блок-схем широко используется во многих важных сферах деятельности: производстве, логистике, системах обслуживания, бизнес-процессах, моделировании компьютерных и телекоммуникационных сетей, и т.д. AnyLogic™ позволяет моделировать при помощи визуальных, гибких, расширяемых, повторно-используемых объектов, как стандартных, так и разработанных Вами. Библиотека Enterprise Library содержит традиционные объекты: очереди, задержки, конвейеры, ресурсы, и т.п., так что модель быстро строится в стиле «перетащить и оставить» (drag-and-drop) и очень гибко параметризуется.

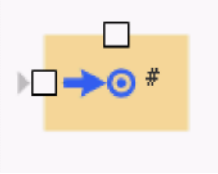
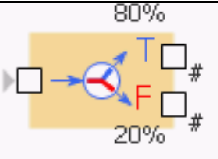
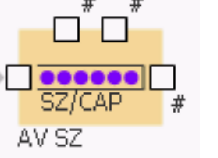
Реализация стандартных объектов Enterprise Library открыта для пользователя, их функциональность может быть как угодно расширена, вплоть до создания собственных библиотек. Чтобы лучше понять, как работают объекты библиотеки, и как создать свои собственные объекты с требуемой функциональностью, Вы можете изучить код объектов библиотеки. Компания XJ Technologies приветствует любые предложения по улучшению функциональности, содержания, анимации и производительности библиотеки. Мы делаем все от нас зависящее, чтобы сделать библиотеку еще более совершенной и удобной для пользователя.

Сборка блок-схемы из объектов

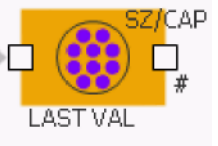
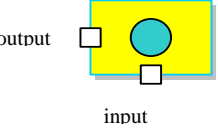

Классы активных объектов библиотеки AnyLogic™ Enterprise Library являются строительными блоками, с помощью которых Вы будете строить блок-схемы. Библиотечный класс сообщений Entity является базовым классом для заявок, ресурсов и транспортеров. В этом справочном руководстве объекты делятся по своей функциональности на шесть категорий, их краткое описание дано в приведенной ниже таблице.

Поток заявок


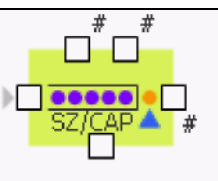
Значок	Имя	Описание
	Source	Генерирует заявки.
	Sink	Удаляет поступающие заявки.
	Enter	Добавляет в блок-схему заявки, созданные в каких-то других блоках

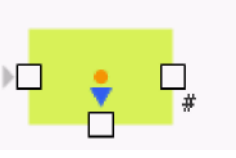
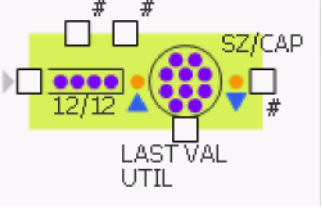
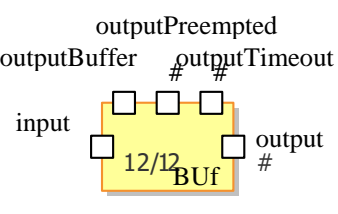
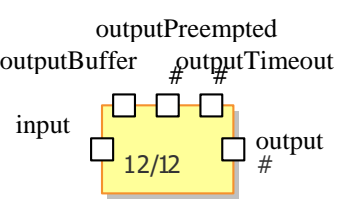
	Exit	Принимает поступающие заявки.
	SelectOutput	В зависимости от заданного условия пересылает заявку на один из выходных портов.
	Queue	Хранит заявки в указанном порядке.

Обработка

Значок	Имя	Описание
	Delay	Задерживает заявки на заданный промежуток времени.
	Gulat	Блок моделирует задержку / «прогулку», после которой заявка может вернуться в систему. Одновременно может «гулять» до 100000 заявок.
	func	Линейно - кусочная функция. Может быть использована как непрерывно-линейная и как дискретная (для задания вероятностей). Задаются точками.

Работа с ресурсами

Значок	Имя	Описание
	Resource	Предоставляет ресурсы, которые могут понадобиться заявкам.
	SeizeQ	Занимает ресурсы, запрошенные заявкой.

	Release	Освобождает ранее занятые заявкой ресурсы.
	ProcessQ	Занимает ресурсы, запрошенные заявкой, задерживает заявку, и затем освобождает ресурсы.
	process	Занимает ресурсы для заявки, задерживает заявку, а затем освобождает занятые ей ресурсы. Объект содержит последовательную комбинацию объектов SeizeQ, Delay, Release, Resource и др.
	process_b	Отличается от блока только тем, что буфер у каждого прибора свой.

Правила пересылки заявок

При построении блок-схем важно понимать, как происходит обмен заявками между активными объектами. Объекты библиотеки Enterprise Library соблюдают четкие правила передачи заявок. Заявки поступают в объект и покидают его через порты. Порт может работать только в одном направлении: или как входной, или как выходной (мы пока не рассматриваем передачу транспортеров и ресурсов в транспортной сети). Входной порт объекта может быть соединен только с выходным портом. Передача заявок выполняется согласно следующему протоколу:

1. Вначале объект, который намеревается передать заявку, посылает уведомление другому объекту.
2. Если принимающий объект соглашается принять заявку, то он пересылает запрос объекту, который намеревается передать заявку.
3. Заявка передается только в ответ на получение запроса. Если запрос прибывает тогда, когда заявка уже покинула передающий объект, то передается null.

Следовательно, заявка никогда не может прибыть в объект или покинуть его без предшествующего этому согласия другого объекта. Этот протокол является надстройкой над протоколом стандартных портов AnyLogic ; он реализован с помощью подклассов класса стандартного порта AnyLogic™: EntityInPort, EntityOutPort и EntityOutPortQueue. Есть еще два других протокола: для обмена ресурсами между объектом Resource и объектами SeizeQ и Release, а также для обмена транспортерами между объектами Node и Segment, но эти протоколы для пользователя библиотеки не столь важны, поэтому мы оставим их без рассмотрения.

Иногда могут возникнуть ситуации, когда объекты не могут принять новые заявки. Поэтому три объекта библиотеки—Queue, Conveyor и Lane (а также объекты, которые заключают в себе какие-то из этих объектов)—позволяют заявке оставаться в объекте и ждать до тех пор, пока она не сможет его покинуть. В том случае, если заявка будет ждать в выходном порте любого другого объекта, то возникнет ошибка. (В то же время,

это не означает, что заявки не могут находиться в буфере: вполне нормальной ситуацией является то, что сразу несколько заявок передаются на выходной порт, поскольку все они могут покинуть объект за нулевое время). Поэтому Вы должны спланировать диаграмму потока заявок таким образом, чтобы заявки всегда могли покинуть объект в том случае, если им запрещено оставаться в объекте. Особняком стоит объект Source, который в случае необходимости может хранить заявки в своем выходном порту — это сделано для упрощения создания моделей. Мы все же рекомендуем отключить эту опцию, чтобы всегда знать во время работы модели, где находятся заявки.

Вы можете создавать различные конфигурации — соединять несколько выходных портов с одним входным портом, несколько входных портов с одним выходным портом, создавать масштабируемые модели, используя иерархию и регулярные структуры объектов — см. Рисунок 1. Важно понимать, как будут себя вести соединения M:1 и 1:M в случае существования нескольких возможных вариантов передачи заявки. В AnyLogic™, если входной порт может принять заявку сразу от нескольких выходных портов, то он поочередно опросит все выходные порты, которые готовы передать заявки. Если выходной порт подсоединен сразу к нескольким входным портам, и готов передать заявку, то он разошлет всем уведомление, и затем перешлет заявку по первому полученному запросу. Если сразу несколько объектов готовы послать свои запросы, то порядок прихода запросов будет случайным, так что заявка будет послана случайно выбранному объекту.

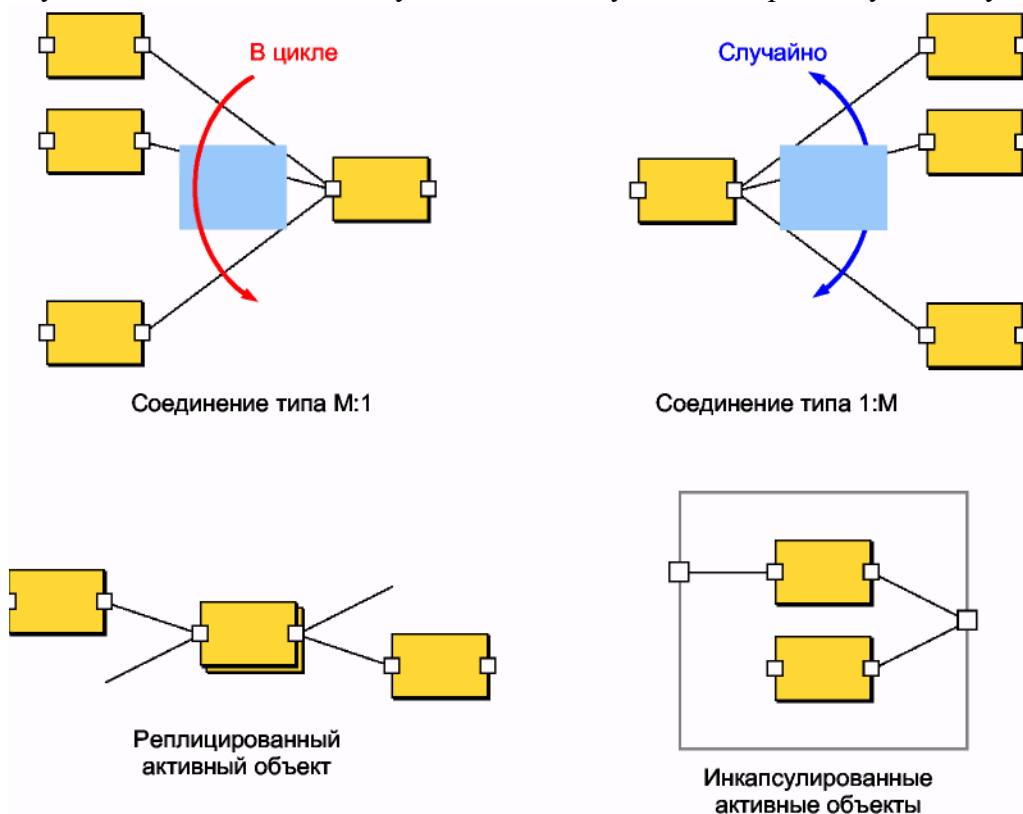


Рисунок 1 Поток заявок и допустимые типы соединений активных объектов

Возможность установления различных типов соединений в AnyLogic™ Enterprise Library имеет большое значение. Поскольку объект может принимать заявку от нескольких источников (которые, в свою очередь могут быть соединены с несколькими получателями), то до самого прихода заявки никогда нельзя сказать с уверенностью, какая заявка будет получена, да и будет ли какая-либо заявка получена вообще. Аналогично, объект никогда не знает о том, получит ли другой объект посланную заявку, до того момента, пока он ее не запросит.

В какой-то момент во время работы модели Вы можете захотеть заблокировать входной порт объекта, чтобы он прекратил принимать заявки. Каждый входной порт каждого объекта Enterprise Library имеет функции `block ()` и `unblock ()`. Вызывая эти функции, Вы можете моделировать рабочие расписания объектов, изменять направления движений заявок, или же накладывать дополнительные ограничения на процесс пересылки заявок.

Динамические параметры

При движении заявок (в том числе ресурсов и транспортеров) по блок-схеме, в определенные моменты (например, при поступлении заявки в объект, освобождении ресурса, сборке заявки, достижении заявкой конца конвейера, и т.д.) может возникнуть необходимость выполнить какие-нибудь действия, например, изменить саму заявку, послать сообщение другим объектам, изменить анимацию, и т.д. Иногда заявка может содержать информацию о том, как объекту следует ее обрабатывать. Например, Вам нужно, чтобы объект Delay задерживал заявку пропорционально значению какого-то поля этой заявки, или присваивал заявкам, поступающим в объект Lane различные скорости, в зависимости от информации, содержащейся в заявке.

AnyLogic™ является чрезвычайно гибким инструментом, позволяющим выполнять всю эту работу с помощью динамических параметров. В AnyLogic™ есть два основных типа параметров активных объектов: простые и динамические. Простые параметры являются чем-то вроде констант внутри объектов, в то время как при каждом вызове динамического параметра происходит исполнение определенного кода. Более того, этот код может быть определен не только при создании класса активного объекта, а и позже, во время создания экземпляра этого класса. Именно поэтому эти параметры и называются динамическими.

Почти все объекты Enterprise Library имеют параметры, тип которых начинается со слова `code`. Это значит, что Вы можете написать последовательность выражений Java (в случае параметра типа `code`) или выражение Java, возвращающее значение типа `t` (в случае параметра типа `code<T>`), прямо в поле задания значения параметра. Этот код будет исполняться при каждом вызове параметра, и пересчитывать заново его значение. Обычно это будет происходить при происхождении каких-то определенных событий в жизненном цикле заявки, поэтому Вам нужно знать, как ссылаться на заявку и переменные объекта. Список переменных каждого объекта дан в описании библиотечных объектов, а текущая заявка в большинстве случаев может быть доступна как `entity`.

Пожалуйста, помните, что код или выражение, заданное в качестве значения динамического параметра, принадлежит сразу двум объектам: инкапсулированному объекту, у которого определен этот параметр, а также объекту, который этот объект содержит (т.е. тому объекту, на чьей структурной диаграмме находится этот объект). Следовательно, если какие-нибудь переменные или функции этих объектов будут называться одинаково, то возникнут конфликты имен. Конфликты разрешаются следующим образом. При вызове переменной или функции из кода динамического параметра, AnyLogic™ вначале интерпретирует ее как принадлежащую инкапсулированному объекту, и уже затем, если имя не было обнаружено — как принадлежащую текущему объекту. Например, если Ваш класс `MyClass` инкапсулирует объект очереди типа `Queue`, то выражение `size()`, заданное, например, в коде параметра очереди `onExit`, будет ссылаться на размер объекта `queue`. Если же Вы имеете функцию `size()` еще и в классе `MyClass` и желаете вызвать ее в поле динамического параметра, то Вы можете написать `MyClass . this . size()`.

Заявки

Базовый класс Entity

Класс `Entity` является базовым классом для всех сообщений, которые посылаются между активными объектами библиотеки Enterprise Library.

Под заявкой в библиотеке Enterprise Library может пониматься:

- заявка в ее обычном понимании (продукт, потребитель, пакет данных, документ),
- ресурс (оператор, машина, критическая секция),
- транспортер (поезд, автобус, корабль, автопогрузчик).

Заявки в их традиционном понимании генерируются объектами `Source`, затем проходят через смоделированную систему, где они обрабатываются, обслуживаются, транспортируются, конкурируют за право обладания ресурсами и, наконец, они эту систему покидают.

Ресурсы, созданные объектами `Resource`, могут быть заняты заявками для выполнения каких-то задач, после чего они освобождаются и возвращаются в объект `Resource`.

Транспортеры, так же как и обычные заявки, создаются объектами Source, затем передаются в объекты Node и используются для транспортировки других заявок между узлами и вдоль сегментов сети.

Объект класса Entity может использоваться в любой из этих ролей, и при необходимости даже менять свою роль во время работы модели.

Заявка может содержать в себе другие заявки, причем уровень вложенности не ограничен. Вложенные заявки хранятся в поле contents типа Vector. Ресурсы, занятые заявкой, хранятся в векторе resources.

Переменные

Тип	Имя	Значение по умолчанию	Описание
int	priority	0	Приоритет заявки (используется объектами Queue). Чем больше значение, тем выше приоритет.
Vector	contents		Содержимое заявки (другие заявки, содержащиеся в этой заявке)
Vector	resources		Ресурсы, которыми владеет данная заявка.
Node	destination		Место назначения заявки, перемещающейся по сети.
double	kol	1	количество попыток занять прибор. Подсчитывается в блоке gulat
double	mem	0	количество памяти, необходимое для решения задачи. Используется в блоке processQ. Значение этого поля необходимо присвоить до попадания заявки в блок processQ.
double	dt1	0	время поступления заявки в очередь. Вычисляется в блоке queue.
double	t_r	0	время появления сообщения (транзакции) в системе. Вычисляется в блоке source.
real	p_real	0	Свободный параметр типа real
integer	p_int	0	Свободный параметр типа integer
Boolean	p_bool	false	Свободный параметр типа boolean
string	p_str		Свободный параметр типа string
ShapeRect	location		Местонахождение заявки в сети*.
Vector	seizedStaff		Вектор занятых ресурсов типа «персонал»*.
Vector	seizedPortable		Вектор занятых переносных ресурсов*.

Vector	seizedStatic		Вектор занятых статических ресурсов*.
Entity	lastResource		Ссылка на последний занятый или освобожденный статический ресурс*.

Переменные, отмеченные * используются при работе с объектами транспортной сети.

Функции

Тип возвращаемого значения	Имя	Описание
void	enableRotation(boolean en)	Разрешает или запрещает вращение анимации заявки.
ShapeBase	getAnimationQ	Возвращает анимационную фигуру заявки.
void	setAnimation(ShapeBase sb)	Задаёт sb в качестве анимационной фигуры заявки.
double	getXQ	Возвращает X-координату анимационной фигуры заявки
double	getYQ	Возвращает Y-координату анимационной фигуры заявки.
void	setX(double x)	Задаёт x в качестве X-координаты анимационной фигуры заявки
void	setY(double y)	Задаёт y в качестве Y-координаты анимационной фигуры заявки
Color	getColorQ	Возвращает цвет анимационной фигуры заявки.
void	setColor(Color c)	Задаёт цвет анимационной фигуры заявки.
void	updateAnimationQ	Перерисовывает заданную по умолчанию фигуру заявки и анимации вложенных заявок.
double	getlnOperatingTimeQ	Возвращает время, в течение которого ресурс был занят: перемещался к запросившим его заявкам, был занят заявками и перемещался вместе с ними, а также возвращался в свое базовое местоположение*.
double	getlnIdleTimeQ	Возвращает время, в течение которого ресурс был свободен*.

int	getCurrentStateQ	Возвращает текущее состояние ресурса*.
int	getResourceTypeQ	Возвращает тип ресурса*.
double	getUtilizationQ	Возвращает показатель использования

Функции, помеченные знаком *, используются только в том случае, если заявка используется в качестве сетевого ресурса (подробную информацию см. в главе 3.7, «Транспортные сети»).

Комментарии

Пожалуйста, обратите внимание, что если Вы работаете с элементом вектора (объекта типа Vector, например contents или resources), получив доступ к нему с помощью функции get (i), то этот объект будет класса Object, и он должен быть приведен к своему классу (Entity или его подклассу).

X- и Y-координаты анимации заявки вычисляются относительно текущего контекста.

Функция updateAnimation () часто переопределяется в подклассах. Это позволяет избавиться от необходимости приведения типов при обновлении анимации.

Создание подклассов класса Entity

Вам может понадобиться определить у заявок дополнительные поля данных, методы или анимацию. В этом случае Вам будет нужно создать новый класс сообщения, например, MyEntity, унаследовать его от класса Entity, и определить необходимые поля и методы в этом классе. Затем, при создании объекта нового класса, Вам будет нужно написать в коде Вашей модели new MyEntity () вместо new Entity (). И наконец, чтобы получить из объекта доступ к созданной Вами функциональности, Вам нужно будет привести заявку к ее типу, например: (MyEntity) entity).myField.

Важным преимуществом AnyLogic™ является то, что Вы можете использовать любые классы Java в качестве заявок, ресурсов и транспортеров: это позволяет делать модели более реалистичными и обеспечивает возможность интеграции моделей AnyLogic™ с другими программными средствами.

Активные объекты

Передача заявок

Source

Источник заявок. Обычно используется в качестве начальной точки потока заявок, или как генератор ресурсов, транспортеров, и т.д. Генерирует заявки любых подклассов базового класса Entity через случайные промежутки времени. Время генерации может как подчиняться закону распределения, так и определяться заданным Вами расписанием. Может быть задано как максимально допустимое число генераций, так и число заявок, создаваемых за каждый раз. Если создаваемые заявки не могут покинуть объект, то они хранятся в буфере выходного порта. Если создание заявок подчиняется закону распределения, то время до создания следующей заявки вычисляется при создании заявки; следовательно, оно может быть сделано вероятностным, детерминированным, зависящим от каких-то дополнительных данных, и т.д.

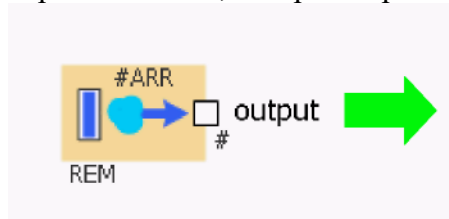


Рисунок 2 Объект Source
Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Функции

Тип возвращаемого значения	Имя	Описание
int	getArrivals()	Возвращает номер последней созданной заявки (первая созданная заявка имеет номер 0).
void	Reset()	Вычисляет заново время до создания следующей заявки.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onExit		Код, выполняемый, когда заявка покидает объект.
Class	newEntity	Entity.class	Тип создаваемой заявки.
	generationType	distribution	Определяет, как источник будет создавать заявки—базируясь на законе распределения (distribution) или согласно расписанию (arrivalList).
double	firstArrivalTime	0	Абсолютное время создания первой заявки. Имеет смысл только тогда, когда создание подчиняется закону распределения. Если выбран тип генерации arrivalList, то время прибытия первой заявки будет соответствовать первому значению в этом списке.
code<double>	interarrivalTime	exponential(1)	Выражение, вычисляющее время до создания следующей заявки, если генерация заявок подчиняется закону распределения (в параметре <i>generationType</i> выбрано distribution).

code<int>	entitiesPerArrival	1	Выражение, вычисляющее число заявок, создающихся за один раз. Если используется расписание <i>arrivalList</i> , то этот параметр игнорируется, а число создаваемых заявок определяется соответствующим значением в таблице.
LookupTable	arrivalList	null	Если заявки создаются согласно расписанию (в поле параметра <i>generationType</i> выбрано <i>arrivalList</i>), то здесь должна быть задана таблица преобразования, задающая времена прибытия заявки, и число заявок, прибывающее в каждый конкретный раз.
int	period	aperiodic	Если используется расписание (<i>arrivalList</i>), то этот параметр задает период прибытия заявок в модельных единицах времени. Параметр не используется, если создание заявок подчиняется закону распределения.
int	arrivalsMax	infinity	Максимальное число генераций.
boolean	canWaitAtOutput	true	Если true, то если заявки не смогут покинуть объект, то они будут храниться в буфере выходного порта (не более 1000), в противном случае возникнет ошибка.

Комментарии

Чтобы создавать заявки подкласса MyClass общего класса Entity, выберите MyClass . class из выпадающего списка в параметре *newEntity*.

Чтобы, например, создать n заявок при инициализации модели, установите *arrivalsMax* в 1 и *entitiesPerArrival* в n.

Если Вы хотите динамически изменять интенсивность потока создания заявок, то можете создать параметр *rateMean* в объекте верхнего уровня (обычно это объект Main); выбрать *distribution* в параметре *generationType*, в качестве значения *interarrivalTime* написать, например, *exponential (rateMean)*; и связать параметр *rateMean* с ползунок. Пожалуйста, обратите Ваше внимание на то, что когда ползунок будет задавать 0 в качестве значения параметра *rateMean*, объект Source не сгенерирует ни одной заявки. Чтобы объект Source снова заработал, нужно будет вызвать его функцию *reset ()* — это произведет перевычисление

значения *interarrivalTime* - времени до следующей генерации. Вы можете сделать это прямо в коде обработчика события ползунка.

Если Вы хотите генерировать заявки в строго определенные моменты времени, то воспользуйтесь режимом создания заявок по расписанию *arrivalList*. Создайте таблицу преобразования с временными метками в качестве аргументов и количеством заявок, генерируемых в эти моменты времени, в качестве значений функции. Выберите в параметре *generationType* значение *arrivalList* и укажите Вашу таблицу преобразований в качестве значения параметра *arrivalList*. Чтобы сделать таблицу периодической, задайте период в параметре *period*. Значения времен создания заявок должны быть отсортированы в возрастающем порядке. Если таблица будет содержать повторяющиеся значения или будет неотсортирована, то поведение объекта будет неопределено.

Если Вы соедините объект *Source* с объектом *Delay* или *Conveyor*, то может возникнуть ситуация, когда сгенерированная заявка не сможет сразу же покинуть объект *Source*. Эту проблему можно решить, поместив между этими объектами объект *Queue*; но чтобы упростить создание моделей, объект *Source* имеет параметр *canWaitAtOutput*, позволяющий заявкам некоторое время храниться в буфере выходного порта. Мы рекомендуем отключить этот режим после первого же успешного запуска Вашей модели, задав *false* в качестве значения параметра.

Sink

Уничтожает поступившие заявки. Обычно используется в качестве конечной точки потока заявок. Объект *Sink* автоматически подсчитывает входящие заявки и высчитывает среднюю интенсивность входящего потока.

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Функции

Тип возвращаемого значения	Имя	Описание
void	Block()	Блокирует входной порт объекта.
void	Unblock()	Разблокировывает входной порт объекта.
boolean	Blocked()	Возвращает true, если входной порт заблокирован, и false — если нет.
int	getCount()	Возвращает число прошедших заявок.
void	Reset()	Обнуляет значение счетчика заявок и производит сброс статистики среднего значения интервала между поступлением заявок.
double	getAvgInterarrivalTime()	Возвращает среднее значение интервала поступления заявок.
double	getAverageRate()	Возвращает среднюю интенсивность входящего потока заявок.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.

Комментарии

Не подсоединенный ни к одному порту выходной порт не сможет произвести вывод заявок, поэтому если Вам нужно удалить заявки, в конце блок-схемы нужно поместить объект Sink или Exit.

Enter

Пересылает заявки, переданные этому объекту либо "явно" через входной порт *inputExternal*, либо с помощью функции объекта *take()*, дальше по блок-схеме. В комбинации с объектом *Exit*, он может быть использован для направления заявок в различные блоки модели без необходимости графического соединения этих блоков на блок-схеме. Также он может служить интерфейсом между потоком заявок и другими частями модели. Пришедшая в объект заявка сразу же его покидает.

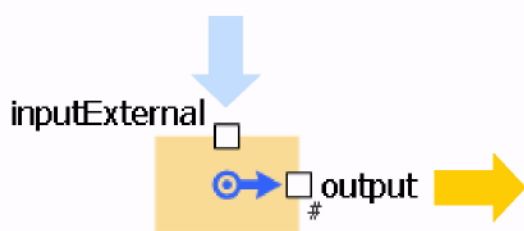


Рисунок 4 Объект Enter
Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Функции

Тип возвращаемого значения	Имя	Описание
void	take(Entity e)	Направляет заявку e в выходной порт <i>output</i> .

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onExit		Код, выполняемый, когда заявка покидает объект.

Комментарии

Объект Enter может использоваться для следующих целей:

- Если заявка, созданная где-то вне потока заявок (в другой части модели,

описанной другими средствами), должна быть добавлена в поток. В этом случае объект Enter играет роль своего рода интерфейсного объекта между различными подходами моделирования. Если заявка появляется на выходе внешнего объекта, то этот порт может быть соединен напрямую к порту *inputExternal*; в противном случае нужно использовать функцию *take ()*. Заметьте, что порт *inputExternal* является обычным портом, не поАдерживающим протокол потока заявок.

- Когда невозможно, или по каким-то причинам нежелательно устанавливать графическое соединение между двумя точками потока заявок; например, когда заявки динамически направляются в зависимости от какого-то условия из одного блока в несколько других. Объект Enter в этом случае используется вместе с объектом Exit.
- Когда заявка, извлеченная из потока заявок средствами API (например, с помощью функции *remove ()* объекта *Queue*), снова добавляется в поток.

Exit

Принимает входящие заявки. Обычно используется в качестве конечной точки потока заявок. В комбинации с объектом Enter, он может быть использован для направления заявок в различные блоки модели без необходимости их графического соединения на блок-схеме. Также объект может служить интерфейсом между потоком заявок и другими частями модели, поскольку при получении заявки Exit «явно» выводит ее через порт *outputExternal*. Пришедшая в объект заявка покидает его мгновенно.

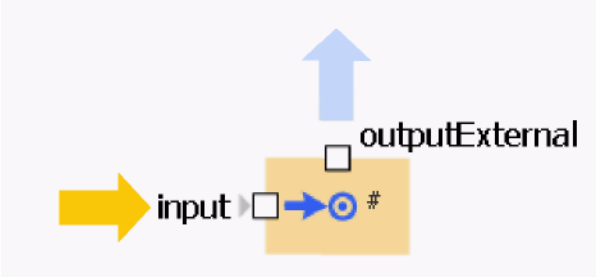


Рисунок 5 Объект Exit

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.
void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.
int	getCountQ	Возвращает число заявок, прошедших через объект.
void	resetQ	Обнуляет значения счетчика входящих заявок и среднего значения интервала времени между поступлением заявок.
double	getAvgInterarrivalTimeQ	Возвращает среднее значение временного интервала между поступлением заявок.

double	getAvgRateQ	Возвращает среднее значение интенсивности входящего потока.
--------	-------------	---

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.

Комментарии

Не подсоединенный ни к какому порту выходной порт не сможет произвести вывод заявок, поэтому если Вам нужно удалить заявки, поместите в конце блок-схемы объект Exit.

Объект Exit может не только уничтожать заявки, но также направлять их в произвольные блоки блок-схемы. Например, чтобы направить заявки в объект Enter, названный myEnter, напишите myEnter. take (entity) в параметре *onEnter* объекта Exit.

Заявка, покидающая поток заявок в объекте Exit, всегда направляется в порт *outputExternal*. Если к нему не подсоединен ни один порт, заявка исчезает. В противном случае она передается в подсоединенные порты. Заметьте, что *outputExternal* является обычным портом, не поддерживающим протокол потока заявок.

SelectOutput

Принимает заявку, и затем, в зависимости от заданного условия, передает ее на один из двух выходных портов. Условие может зависеть от самой заявки или от какой-то другой информации. Поступившая заявка покидает объект в тот же момент времени.

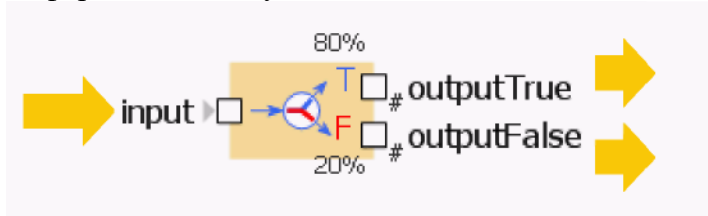


Рисунок 11 Объект SelectOutput

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.
void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.
code	onExitTrue		Код, выполняемый, когда заявка покидает объект через порт <i>outputTrue</i> .
code	onExitFalse		Код, выполняемый, когда заявка покидает объект через порт <i>outputFalse</i> .
code<boolean>	selectCondition	Uniform()<0.5	Условие, вычисляемое отдельно для каждой заявки. Если результат равен true, то заявка покидает объект через порт <i>outputTrue</i> , иначе - через <i>outputFalse</i> . По умолчанию выбрано <code>uniform() < 0.5</code> , т.е. заявки будут покидать объект через разные порты с равной вероятностью.

Комментарии

Значение, выбранное в параметре *selectCondition* по умолчанию, является примером вероятностного распределения заявок. Вы можете применить и детерминированную сортировку заявок. Предположим, что Вы хотите сортировать заявки типа *MyEntity* по разным портам в зависимости от значения их целочисленного поля *size*. Тогда Вы можете написать в коде параметра *selectCondition*: `((MyEntity) entity) . size>98`.

Выбор может также зависеть, например, от состояния другого объекта. Например, Вы хотите направлять заявки в порт *OutputTrue*, только если объект *queue* типа *Queue* не заполнен. Тогда Вы пишете: `queue.canEnter()`.

Объект *SelectOutput* обычно используется для сортировки заявок в зависимости от их типов. Например, Вы можете написать в параметре *selectCondition* `entity instanceof Cat`, тогда объекты типа *Cat* будут покидать объект через порт *outputTrue*, а все остальные - через *outputFalse*.

Queue

Объект *Queue* моделирует очередь, он хранит поступающие заявки в определенном порядке: FIFO (заявки помещаются в очередь в порядке поступления), LIFO (заявки помещаются в порядке, обратном поступлению), RANDOM (заявки помещаются в произвольные места очереди) или PRIORITY (заявки помещаются в очередь в соответствии со значением своих полей *priority*). Заявка может покинуть объект *Queue* различными способами:

- "обычным способом" через порт *output*, когда объект, следующий в блок-схеме за этим объектом, готов принять заявку
- через порт *outputTimeout*, после того, как заявка проведет в очереди заданное количество времени (если включен режим таймаута)
- через порт *outputPreempted*, будучи вытесненной другой поступившей заявкой при заполненной очереди (если включен режим вытеснения)
- "вручную", путем вызова функции `remove (int i)`

В первом случае объект *Queue* покидает заявка, находящаяся в самом начале очереди (в нулевой позиции). Если заявка направлена в порт *outputTimeout* или *outputPreempted*, то она должна покинуть объект мгновенно. Если включена опция вытеснения *preemption*, то объект

Queue всегда готов принять новую заявку, в противном случае при заполненной очереди заявка принята не будет.

Вместимость очереди может быть установлена бесконечной (infinity). Если Вы хотите отображать на анимации не все, а только некоторые заявки, то используйте параметр *entitiesToAnimate*. Этот параметр задает, сколько заявок от начала очереди должно быть отображено на анимации. Аниматор должен иметь возможность показывать заданное количество заявок. Например, Вы можете использовать аниматор типа SET с ломаной линией, выбранной в качестве анимационной фигуры и значением параметра *entitiesToAnimate*: number of points in polyline.

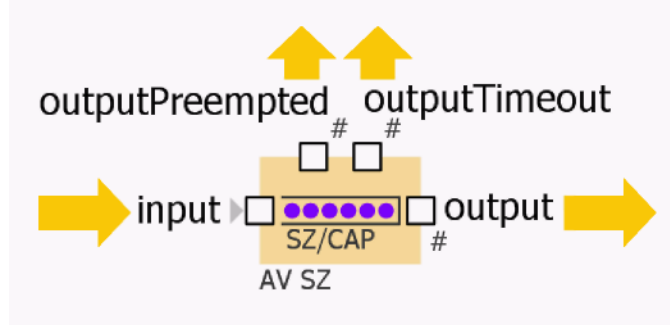


Рисунок 12 Объект Queue

Переменные

Тип	Имя	Описание
Entity	Entity	Текущая заявка.
int	Position	Позиция очереди, в которую помещена только что прибывшая заявка.
double	timeoutValue	Значение таймаута, установленное для только что прибывшей заявки.
Entity	get(int i)	Возвращает заявку из i-й позиции (ближайшей к выходу считается позиция 0).
Entity	remove(int i)	Удаляет заявку из i-й позиции и возвращает ее.
boolean	canEnterQ	Возвращает true, если новая заявка может быть помещена в очередь. Если включен режим вытеснения, то это происходит всегда; иначе это эквивалентно size () < capacity.
TimedDataSet	getStatsSizeQ	Возвращает статистику размера очереди.
void	resetStats Q	Производит сброс накопленной статистики.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.

void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.
int	sizeQ	Количество заявок в очереди.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.
code	onExitPreempted		Код, выполняемый, когда заявка покидает объект через порт <i>outputPreempted</i> , будучи вытесненной другой заявкой.
code	onExitTimeout		Код, выполняемый, когда заявка покидает объект через порт <i>outputTimeout</i> в результате истечения таймаута ее пребывания в очереди.
code	onAtExit		Код, выполняемый, когда заявка перемещается в нулевую позицию очереди и готова покинуть объект.
code	onExit		Код, выполняемый, когда заявка покидает объект или через порт <i>output</i> , или после вызова функции <i>remove ()</i> .
int	queueType	FIFO	Тип очереди. Может быть одним из следующих: FIFO LIFO RANDOM PRIORITY
integer	capacity	100	Вместимость очереди.
integer	entitiesToAnimate	all	Количество заявок, которое будет отображено на анимации.
boolean	preemption	false	Если true, то включен режим вытеснения.
boolean	timeout	false	Если true, то включен режим таймаута.

code<double>	timeoutTime	infinity	Выражение, вычисляющее значение таймаута для заявки. Применяется только при включенном режиме таймаута (<i>timeout</i> равен true).
boolean	statsEnabled	false	Если true, то для этого объекта будет собираться статистика.
ShapeBase	animationShape		Шаблон анимации очереди. Тип зависит от выбранного аниматора.
int	animationType	AUTO	Тип аниматора очереди. Может быть одним из следующих: AUTO SINGLE SET BAG ARRANGED QUEUE
boolean	animationForward	true	Направление движения заявок на анимации, если объект отображается на анимации ломаной линией.
Набор данных	lang		длина очереди
Набор данных	ddt		время нахождения заявки в очереди

Комментарии

К моменту выполнения кода обработчика события *onEnter* заявка уже будет помещена в очередь, и будут известны номер ее позиции в очереди *position* и таймаут *timeoutvalue*. К моменту выполнения кода обработчиков событий *onExit*, *onExitPreempted* или *onExitTimeout* заявка уже будет вынута из очереди; в последнем случае номер последней позиции, которую она занимала в очереди, будет доступен как *position*.

Вытеснение заявок работает следующим образом. Поступающая заявка всегда принимается объектом и помещается в очередь согласно типу очереди. Если очередь переполняется, то последняя в очереди заявка (это может быть и только что пришедшая заявка) удаляется и покидает объект через порт *outputPreempted*.

Если включен режим таймаута, то для каждой заявки, помещенной в очередь, запускается таймер. Значение таймаута вычисляется для каждой заявки отдельно, в соответствии со значением выражения *timeoutTime*; если возвращается 0, то таймер не создается. Если таймаут истекает до того, как заявка покидает очередь, то она вынимается из очереди и направляется в порт *outputTimeout*.

При типе очереди RANDOM прибывающая заявка помещается в случайно выбранную позицию очереди. Значение поля *priority* заявки *entity* учитывается только объектом с типом очереди PRIORITY. Чем больше значение этого поля, тем выше приоритет заявки.

Вместимость очереди может быть изменена динамически.

Delay

Задерживает заявки на заданное время. Одновременно могут быть задержаны сразу несколько заявок (не более заданной вместимости объекта *capacity*). В отличие от объекта *Server*, заявки задерживаются независимо друг от друга — время задержки вычисляется отдельно для

каждой заявки. Как только время задержки истекает, заявка тут же покидает объект. Если объект Delay заполнен полностью, то новую заявку он не примет.

Вместимость объектов Delay может изменяться с помощью объекта Schedule. Объект Schedule автоматически управляет вместимостью в соответствии с заданными значениями времен до следующей поломки и до починки (TTF и TTR) и рабочим расписанием.

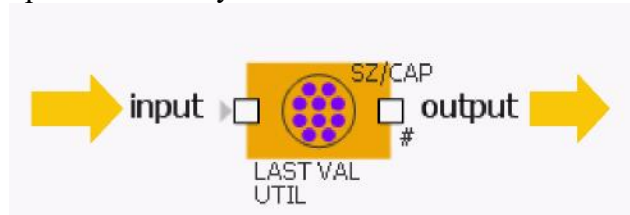


Рисунок 21 Объект Delay

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.
double	delayTime Value	Значение задержки для текущей заявки.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.
void	resetStats Q	Сбрасывает статистику, собранную объектом.
void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.
int	sizeQ	Число задержанных (находящихся в объекте) в данный момент заявок.
Entity	get(inti)	Возвращает i-ю заявку.
boolean	canEnterQ	Возвращает true, если новая заявка может быть принята, т.е. если вместимость объекта Delay еще не достигнута.
TimedDataSet	getStatsUtilizationO	Возвращает статистику использования объекта.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.
code	onExit		Код, выполняемый, когда заявка покидает объект.

code<double>	delayTime	triangular(0.5, 1, 1.5)	Выражение, вычисляющее время задержки для текущей заявки.
double	scale	1	Если в качестве значения параметра <i>delayTime</i> выбрано <i>length of polyline</i> , то время задержки будет равно длине анимационной фигуры объекта (ломаной линии), умноженной на этот коэффициент. Поэтому этот коэффициент часто выбирается равным $1/\text{speed}$.
int	capacity	1	Вместимость объекта.
boolean	statsEnabled	false	Если true, то для объекта собирается статистика, если false, то нет.
ShapeBase	animationShape		Шаблон анимации объекта. Тип зависит от аниматора.
int	animationType	AUTO	Тип аниматора. Может быть одним из следующих: AUTO SINGLE SET BAG ARRANGED MOVEMENT
boolean	animationForward	true	Задаёт направление движения заявок на анимации в том случае, если объект отображается ломаной линией.
Schedule	schedule	without_schedule	Имя объекта, задающего расписание изменения вместимости объекта.
double	koef		коэффициент занятости прибора
double	sum		суммарное время занятости прибора
Набор данных	win		окна приборов

Комментарии

Когда выполняется код параметра *onEnter*, значение времени, на которое должна быть задержана заявка, доступна как *delayTimeValue*.

Время задержки может быть стохастическим (как, например, значение по умолчанию), детерминированным, может зависеть от заявки или любой другой информации. Если,

предположим, Вы хотите задерживать заявки типа Packet на время, пропорциональное значению поля size, тогда Вам нужно написать:

`((Packet)entity).size*k.`

Вместимость объекта Delay может быть изменена динамически путем вызова функции `set_capacity()`, или с помощью объекта Schedule. Один объект Schedule может управлять вместимостями сразу нескольких объектов Delay. Если вместимость была уменьшена до значения, меньшего, чем число заявок, находящихся в объекте в данный момент времени, то эти заявки останутся в объекте Delay до истечения своих времен задержек. Поэтому функция `size()` (возвращающая число задерживаемых объектом заявок в данный момент времени) в этом случае будет возвращать значение, большее

вместимости *capacity* объекта. Объект примет новые заявки только после того, как лишние заявки покинут объект, и `size()` снова будет меньше *capacity*.

Работа с ресурсами

Resource

Предоставляет ресурсы, которые могут быть заняты и освобождены заявками с помощью объектов SeizeQ, ProcessQ и Release. Ресурсы — это объекты класса Entity или созданного Вами подкласса этого класса. Объект Resource может создавать, хранить, выдавать и забирать ресурсы. Количество ресурсов может изменяться динамически. Объект Resource лучше всего подходит для моделирования относительно небольших количеств индивидуально неповторимых предметов, таких, как операторы, машины, устройства, критические секции, и т.п.

В любой момент времени ресурсом может владеть только одна заявка; следовательно, заявки конкурируют за право обладания ресурсами. В случае одновременного получения нескольких запросов, Resource удовлетворяет их вначале в соответствии с их приоритетами, а затем — согласно их временным меткам. Приоритетом в данном случае является значение параметра *priority* объекта SeizeQ, который пытается занять ресурс для заявки (не путайте с полем *priority* заявки). Временная метка хранит значение времени, с которого заявка ожидает получения ресурса (время, когда заявка попала в очередь объекта SeizeQ). Более старые запросы обслуживаются в первую очередь. В том случае, если запрос не может быть обслужен из-за нехватки свободных ресурсов, он пропускается, а обслуживаются запросы с более низкими приоритетами или с более поздней временной меткой.

Объект Resource имеет порт *access*, который должен быть подсоединен к портам *access* объектов SeizeQ, Release или ProcessQ. Один объект Resource может быть подсоединен сразу к нескольким таким объектам, и наоборот, один объект SeizeQ или Release может быть подсоединен сразу к нескольким объектам Resource.



Рисунок 23 Объект Resource

Переменные

Тип	Имя	Описание
Entity	unit	Текущий ресурс.

Функции

Тип возвращаемого значения	Имя	Описание
int	Size()	Количество свободных ресурсов.
Entity	get(inti)	Возвращает i-й ресурс.
int	getSeized()	Возвращает количество занятых ресурсов.
TimedDataSet	getStatsUtilization()	Возвращает статистику использования ресурса.
void	resetStats()	Производит сброс статистики использования ресурса.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onSeizeUnit		Код, выполняемый при занятии ресурса.
code	onReleaseUnit		Код, выполняемый при освобождении ресурса.
code	onGenerate		Код, выполняемый при создании нового ресурса.
int	capacity	1	Количество ресурсов.
code<Entity>	newUnit	Entity.class	Тип ресурса.
boolean	statsEnabled	false	Если true, то для объекта собирается статистика, если false, то нет.
ShapeBase	animationShape		Шаблон анимации ресурса. Тип зависит от аниматора.
int	animationType	AUTO	Тип аниматора очереди. Может быть одним из следующих: AUTO SINGLE SET BAG ARRANGED

Комментарии

При начале работы модели объект Resource создает столько ресурсов типа, заданного в параметре *newUnit*, сколько задано параметром *capacity*. После создания каждого нового ресурса выполняется код, заданный в параметре *onGenerate* (в этом коде созданный ресурс может быть доступен как unit). Количество ресурсов может изменяться динамически. Если значение параметра *capacity* увеличивается, то создаются дополнительные ресурсы. Если оно уменьшается, то соответствующее число свободных ресурсов удаляется; если требуемого числа свободных ресурсов нет, то объект Resource ждет, пока не освободится необходимое

для удаления количество

ресурсов. В конечном счете, число ресурсов будет снова равно значению *capacity*.

Код параметров *onSeizeUnit* и *onReleaseUnit* выполняется для каждого занимаемого и освобождаемого ресурса; т.е., если занимаются 5 ресурсов, то код параметра *onSeizeUnit* выполняется 5 раз. К моменту выполнения кода *onSeizeUnit* ресурсы уже будут удалены из объекта *Resource*. Когда выполняется код *onReleaseUnit*, ресурсы уже добавлены обратно в объект.

Занятые ресурсы не удаляются из аниматора объекта *Resource*, как это делается в большинстве случаев, Вам нужно просто изменить их внешний вид, чтобы показать, что они заняты.

SeizeQ

Занимает для заявки заданное количество ресурсов определенного типа. Содержит очередь, в которой заявки ожидают, пока запрашиваемый ресурс не станет доступным.

Вначале ресурс запрашивается для первой заявки из очереди, и пока эта заявка владеет ресурсом, ресурсы для последующих заявок не выделяются (даже если они и могли бы быть выделены). Запрос содержит параметр *priority* объекта *SeizeQ* и временную метку заявки — время, когда заявка поступила в объект *SeizeQ*. Как только для заявки занимается ресурс, она немедленно покидает объект. К порту *access* объекта *SeizeQ* могут быть подсоединены как один, так и несколько объектов *Resource*. Объект *Resource*, так же как и количество ресурсов, выбирается отдельно для каждой заявки. Функциональность и интерфейс внутренней очереди *Queue*, в том числе режимы вытеснения и таймаута, полностью унаследовываются объектом *SeizeQ*.

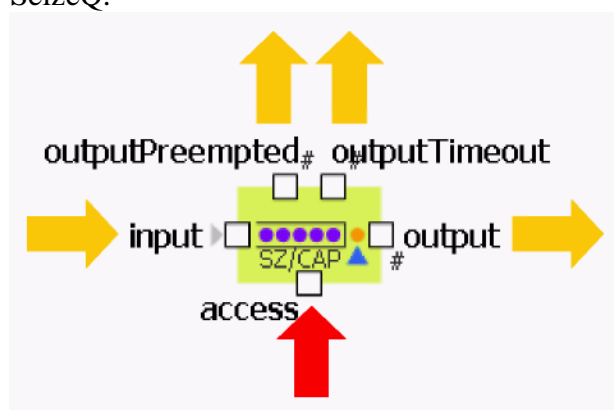


Рисунок 24 Объект *SeizeQ*

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.
Resource	resource	Выбранный объект <i>Resource</i> .
int	quantityValue	Количество ресурсов, запрашиваемых текущей заявкой.
Entity	unit	Только что занятый ресурс.
int	position	Переменные очереди, см. <i>Queue</i> .
double	timeoutValue	
double	all	количество прошедших через блок заявок.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.
void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.
int	sizeQ	Функции очереди, см. Queue.
Entity	get(inti)	
Entity	remove(int i)	
boolean	canEnterQ	
TimedDataSet	getStatsSizeQ	
void	resetStats 0	

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Параметры очереди, см. Queue.
code	onExitPreempted		
code	onExitTimeout		
int	queueType	FIFO	
int	capacity		
int	entitiesToAnimate	all	
boolean	preemption	false	
boolean	timeout	false	
code<double>	timeoutTime	infinity	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
boolean	statsEnabled	false	
code	onSeizeUnit		Код, выполняемый, когда ресурс занимается заявкой.
code	onExit		Код, выполняемый, когда заявка покидает объект.

code<int>	quantity	1	Выражение, вычисляющее количество ресурсов, которое необходимо получить заявке.
code<Resource>	selectResource		Выражение, выбирающее объект Resource. Если возвращает null, то объект выбирается случайно.
int	priority	0	Приоритет объекта. Чем больше значение, тем выше приоритет.
Boolean	popitka	false	собирать статистику количества попыток заявки занять прибор.
Набор данных	kolp		количество попыток занятия прибора

Комментарии

Если к объекту *SeizeQ* не подсоединен ни один объект *Resource*, или если к нему не подсоединен объект *Resource*, выбранный в параметре *selectResource*, то при попытке заявки занять ресурс возникнет ошибка. Если *selectResource* возвращает null, то объект *Resource* случайно выбирается из всех подсоединенных объектов *Resource*.

Когда новая заявка ставится в нулевую позицию очереди, вычисляется необходимое число ресурсов, и объект *Resource* выбирается путем вычисления параметров *quantity* и *selectResource* — именно в таком порядке. Поскольку в зависимости от типа очереди заявка может быть сдвинута другой заявкой, и затем вновь попасть на нулевую позицию очереди, эта процедура может повторяться несколько раз для одной и той же заявки.

Код параметра *onSeizeUnit* выполняется для каждого занятого ресурса; т.е., если будет занято 5 ресурсов, то *onSeizeUnit* выполнится 5 раз. К моменту вызова кода параметра *onSeizeUnit* ресурс уже будет добавлен к вектору *resources* этой заявки. К моменту вызова кода *onExit* заявка уже будет удалена из очереди.

Release

Освобождает ресурсы, ранее занятые заявкой. Вы можете либо указать количество ресурсов одного типа, либо задать условие для их выбора. К порту *access* объекта *Release* могут быть подсоединены один или сразу несколько объектов *Resource*, поэтому для каждой заявки может быть выбран ресурс своего типа. Процедура освобождения ресурсов занимает нулевое время, так что заявка покидает объект, как только она в него поступила.

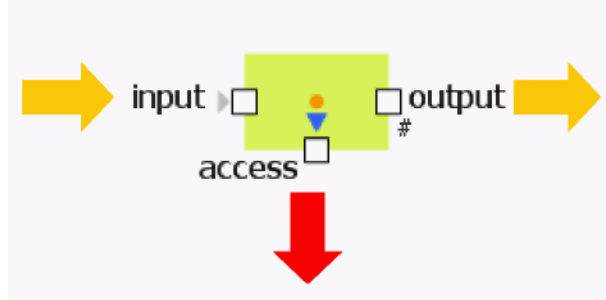


Рисунок 25 Объект Release

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.
Resource	resource	Выбранный объект Resource.
int	quantityValue	Количество ресурсов, занятых заявкой.
Entity	unit	Ресурс, занятый или проверяемый в данный момент.

Функции

Тип возвращаемого значения	Имя	Описание
void	blockQ	Блокирует входной порт.
void	unblockQ	Разблокировывает входной порт.
boolean	blockedQ	Возвращает true, если входной порт заблокирован, и false — если нет.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.
code	onReleaseUnit		Код, выполняемый, когда ресурс освобождается заявкой.
code	onExit		Код, выполняемый, когда заявка покидает объект.
code<int>	quantity	1	Выражение, вычисляющее количество ресурсов, которые должны быть освобождены.
code<Resource>	selectResource	random	Выражение, вычисляющее, какой объект Resource должен быть выбран. Если возвращает null, то объект выбирается случайно из всех подсоединенных объектов.
code<boolean>	selectUnit	true	Условие, вычисляемое для каждого занятого ресурса и определяющее, должен ли он быть освобожден.

Комментарии

Если к этому объекту не подсоединен ни один объект Resource, или если к нему не подсоединен объект Resource, выбранный в параметре *selectResource*, то при попытке освободить ресурсы возникнет ошибка. Если параметр *selectResource* вернет значение null, то объект Resource будет случайно выбран среди подсоединенных объектов. Например,

предположим, что Вы хотите освободить ресурс типа Operator, тогда Вам нужно написать в поле параметра *selectResource*: unit instanceof Operator.

Код параметра *selectUnit* выполняется для каждого ресурса, занятого заявкой, и в том случае, если код возвращает true, ресурс освобождается. Количество освобождаемых ресурсов ограничивается значением параметра *quantity*.

Код *onReleaseUnit* выполняется для каждого освобождаемого ресурса; т.е., если освобождаются 5 ресурсов, то *onReleaseUnit* вызывается 5 раз. На момент вызова *onReleaseUnit*, ресурс уже удален из вектора resources заявки.

ProcessQ

Занимает ресурсы для заявки, задерживает заявку, а затем освобождает занятые ею ресурсы. К порту *access* объекта ProcessQ должны быть подсоединены один или несколько объектов Resource. Объект содержит последовательную комбинацию объектов SeizeQ, Delay и Release. Функциональность и интерфейс этих трех объектов наследуются объектом ProcessQ (за исключением объекта Release, который настроен так, чтобы освободить именно те ресурсы, которые были ранее заняты объектом SeizeQ).

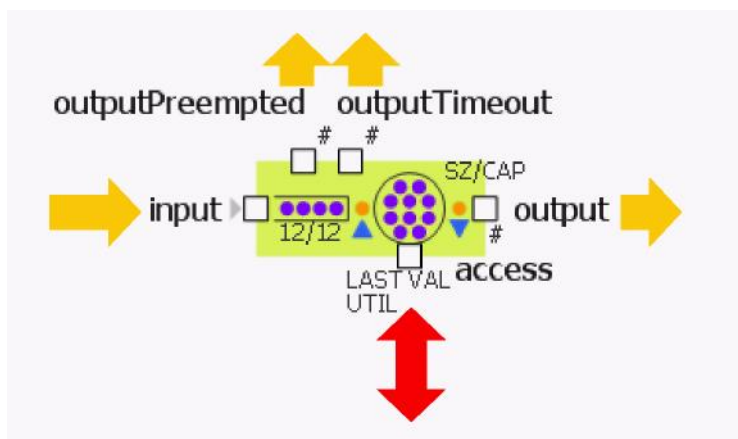


Рисунок 26 Объект ProcessQ

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.
Resource	resource	Выбранный объект Resource.
int	quantityValue	Количество ресурсов, запрашиваемых заявкой
Entity	unit	Ресурс, который занимает или освобождается в данный момент времени.
int	position	Переменные очереди, см. Queue.
double	timeoutValue	
double	delayTime Value	Переменная объекта задержки, см. Delay.

Функции

Тип возвращаемого значения	Имя	Описание
void	Block()	Блокирует входной порт.
void	Unblock()	Разблокировывает входной порт.

boolean	Blocked()	Возвращает true, если входной порт заблокирован, и false —
int	sizeQ()	Функции очереди, см. Queue.
Entity	getQ(inti)	
Entity	removeQ(int i)	
boolean	canEnterQ	
TimedDataSet	getStatsSizeQ()	
int	sizeDelay()	Функции объекта задержки, см. Delay.
Entity	getDelay(int i)	
TimedDataSet	getStatsUtilization()	
void	resetS tats()	Производит сброс статистики, собираемой объектом.

Параметры

Тип	Имя	Значение по Описанию умолчанию	
Code	onEnter		Параметры внутреннего объекта, занимающего ресурсы, см. SeizeQ.
Code	onExitPreempted		
Code	onExitTimeout		
Int	queueType	FIFO	
Int	queueCapacity		
Boolean	preemption	false	
Boolean	timeout	false	
code<double>	timeoutTime	0	
ShapeBase	animationShapeQ		
Int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
code	onSeizeUnit		
code<int>	quantity	1	
code<Resource>	selectResource	пиП	
int	priority	0	
code<double>	delayTime	exponential(1)	Параметры объекта задержки, см. Delay.

int	delayCapacity	100	
ShapeBase	animationShapeDelay		
int	animationTypeDelay	AUTO	
boolean	animationForwardDelay	true	
code	onReleaseUnit		Параметр внутреннего объекта, освобождающего ресурсы, см. Release.
code	onEnterDelay		Код, выполняемый, когда заявка уже заняла необходимый ресурс и поступила в объект задержки.
code	onExit		Код, выполняемый, когда заявка покидает объект.
boolean	statsEnabled	false	Если true, то для этого объекта статистика собирается, иначе - нет.
Boolean	popitka	False	собирать статистику количества попыток заявки занять прибор.

Комментарии

В том случае, если Вы хотите произвести какое-то действие уже после того, как заявка была занята, но до того, как началась ее задержка, Вы можете воспользоваться параметром *OnEnterDelay*. Он является не чем иным, как параметром *onEnter* внутреннего объекта задержки Delay.

ProcessQ_b

Блок отличается от блока ProcessQ только тем, что обладает буфером.

Дополнительные данные дополнительные данные по сравнению с ProcessQ:

Переменные

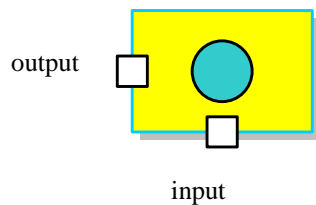
Тип	Имя	Описание
double	buf	Текущий буфер.

Параметры

Тип	Имя	Значение по умолчанию	Описание
double	bufer	0	Размер буфера

Gulat

Блок моделирует задержку / «прогулку», после которой заявка может вернуться в систему. Одновременно может «гулять» до 100000 заявок.



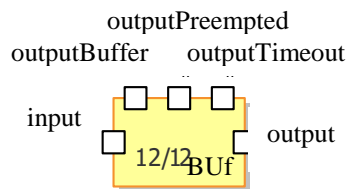
Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Параметры

Тип	Имя	Значение по умолчанию	Описание
Real	Time	triangular(0.5, 1, 1.5)	Выражение, вычисляющее время задержки заявки.

Process



Занимает ресурсы для заявки, задерживает заявку, а затем освобождает занятые ею ресурсы. Объект содержит последовательную комбинацию объектов SeizeQ, Delay, Release, Resource и др. Объект моделирует прибор/приборы (коэффициент *kolichество* определяет количество приборов, моделируемых блоком), занимаемые заявками по некому правилу (параметр *vibor*), с общим буфером (параметр *bufer*), если размера буфера недостаточно, то заявка подается на выход *outputBuffer*.

Переменные

Тип	Имя	Описание
Boolean	frei	Если true, есть свободный прибор, иначе – все заняты
Integer	n	Номер выбранного прибора
double	buf	свободное количество памяти (буфера).

Параметры

Тип	Имя	Значение по умолчанию	Описание

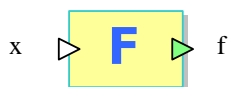
code	onExit		Код, выполняемый, когда заявка покидает объект.
double	bufer	0	размер буфера
Boolean	popitka	False	Если true, то собирать статистику количества попыток заявки занять прибор.
integer	vibor	min koef	Определяет правило занятия прибора. Может быть: min koef (минимальный коэффициент использования); 1 svobodnii (первый свободный); min sadach(минимальное число решенных прибором задач); min queue(минимальная длина очереди)
Integer	kolichestvo	1	Количество приборов
Boolean	Vid	true	Если vid=true, то заявка при отсутствии свободных приборов или нехватки ресурсов (буфера) встает в очередь перед выбранным прибором, иначе покидает блок через порт outputBuffer

Process_b

Блок отличается от блока Process только тем, что буфер в каскаде не общий, а свой у каждого прибора.

Func

Линейно - кусочная функция. Может быть использована как непрерывно-линейная и как дискретная (для задания вероятностей). Задается точками.

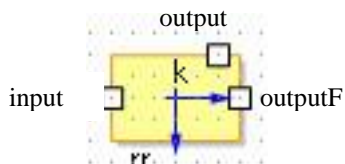


Параметры

Тип	Имя	Значение по умолчанию	Описание
Integer	kol	1	Определяет количество отрезков функции.

Real	xi		Задаёт абсциссу i-точки
real	yi		Задаёт ординату i-точки

Prerivanie



Прибор с прерыванием, задерживает заявку на определенный период времени, если происходит прерывание, то заявка ставится в начало очереди. Через выход output выходят вытесненные заявки, через outputF – обработанные.

Переменные

Тип	Имя	Описание
Entity	entity	Текущая заявка.

Параметры

Тип	Имя	Значение по умолчанию	Описание
code	onEnter		Код, выполняемый, когда заявка поступает в объект.
code	onExit		Код, выполняемый, когда заявка покидает объект.
integer	capacity	1	Вместимость объекта.
boolean	statsEnabled	false	Если true, то для объекта собирается статистика, если false, то нет.
ShapeBase	animationShape		Шаблон анимации объекта. Тип зависит от аниматора.
int	animationType	AUTO	Тип аниматора очереди. Может быть одним из следующих: AUTO SINGLE SET BAG ARRANGED
boolean	animationForward	true	Задаёт направление движения заявок на анимации в том случае, если объект отображается ломаной линией.

boolean	vid	True	Если параметр равен true, то вытесненная заявка обрабатывается с места останова (доделывается), если false -начинает обрабатываться с начала.
boolean	vitesnenie	True	Если параметр выбран true - заявка выходит из блока и ее следует направить в конец очереди или другой блок по условию задачи, иначе, если параметр принимает значение false - заявка переходит в начало очереди.
double	Time	triangular(0.5, 1, 1.5)	Выражение, вычисляющее время задержки для текущей заявки.