

Анализ и проектирование на UML

Новиков Федор Александрович

fedornovikov@rambler.ru

Курс подготовлен по заказу
ООО Сан Майкросистемс СПб

Часть 5

Курс подготовлен при поддержке Sun Microsystems
Правила использования материалов опубликованы на www.sun.ru

План лекций

- Введение в UML
- Обзор языка
- Моделирование использования
- Моделирование структуры
- ✓ Моделирование поведения
- Управление моделями
- Тенденции развития языка
- UML и процесс разработки

5. Моделирование поведения

- 5.1. Конечные автоматы
- 5.2. Диаграммы состояний
- 5.3. Диаграммы деятельности
- 5.4. Диаграммы взаимодействия
- 5.5. Диаграммы коммуникации
- 5.6. Диаграммы последовательности
- 5.7. Моделирование параллелизма

5.1. Конечные автоматы

- Входной алфавит: $A = \{a_1, \dots, a_n\}$
- Алфавит состояний: $Q = \{q_1, \dots, q_m\}$
- Функция переходов: $\delta : A \times Q \rightarrow Q$
- Выходной алфавит: $B = \{b_1, \dots, b_k\}$
- Функция выходов: $\lambda : A \times Q \rightarrow B$
- Автомат Мили: $\lambda : A \times Q \rightarrow B$
- Автомат Мура: $\mu : Q \rightarrow B$
- Начальное состояние: q_1
- Заключительные состояния: $q_i, q_j,$

Модификации конечных автоматов

- **Функция переходов с побочным эффектом = процедура реакции**
- **Реакция: запись/чтение на потенциально бесконечную ленту = машина Тьюринга**
- **Реакция: push, pop (стековая память) = магазинный автомат**
- **Выход одного на входы других: взаимодействующие автоматы**
- **Состояния одного являются входами другого: взаимодействующие автоматы**

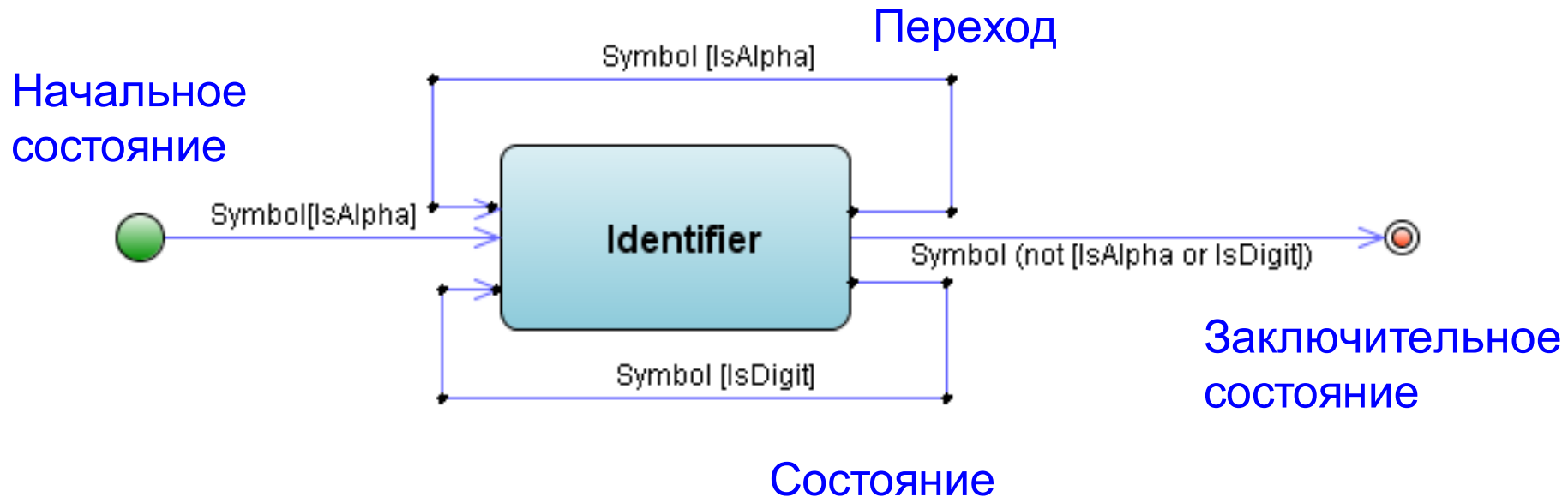
Почему автоматы интересны?

- Теорема Райса. Все нетривиальные свойства программ алгоритмически неразрешимы
 - Эквивалентность
 - Завершаемость
 - Оптимальность
 - Тотальность
 - ...
- Известны частные случаи (автоматы), для которых есть разрешающие алгоритмы

Таблица конечного автомата

	Стимул1	...	СтимулN
Состояние1			
...		РеакцияK СостояниеL	
СостояниеM			

Граф [состояний и] переходов



Применения автоматов

- Трансляция / интерпретация
- Пассивный пользовательский интерфейс
- Программы, управляемые событиями
- Моделирование жизненного цикла объекта
 - Если поведение зависит от прошлого
 - Если есть асинхронные стимулы
- Спецификация множества допустимых последовательностей вызовов операций (*протокол*)

Программирование конечных автоматов

- **state := 1**
- **while state ≠ k**
- **stimulus := get()**
- **switch state**
- **case 1**
- **switch stimulus**
- **case A**
- **...**
- **...**
- **...**

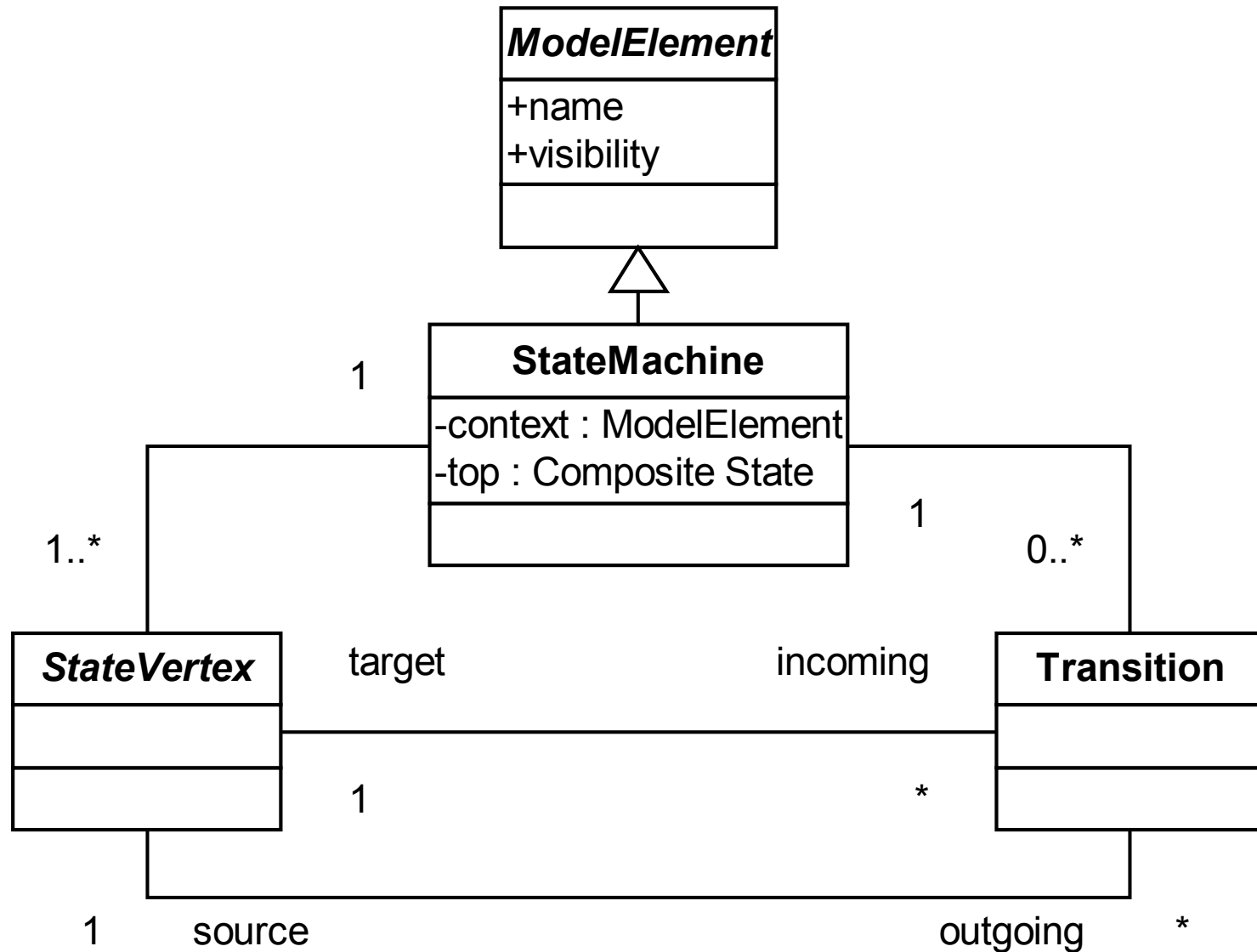
Автоматное программирование

- UniMod Гурова (2005)
- SWITCH-технология Шалыто (1991)
 - <http://is.ifmo.ru>
- Р-технология Вельбицкого (1976)
- ...
- Предопределенная архитектура \approx универсальный образец проектирования
- Источники событий → Управляющие автоматы → Объекты управления

5.2. Диаграммы состояний

- **Состояния (state)**
 - Простые (simple)
 - Составные (composite)
 - Специальные (pseudo)
- **Переходы (transition)**
 - Начало (source)
 - Событие (event или trigger)
 - Сторожевое условие (guard condition)
 - Действие (action или effect)
 - Конец (target)

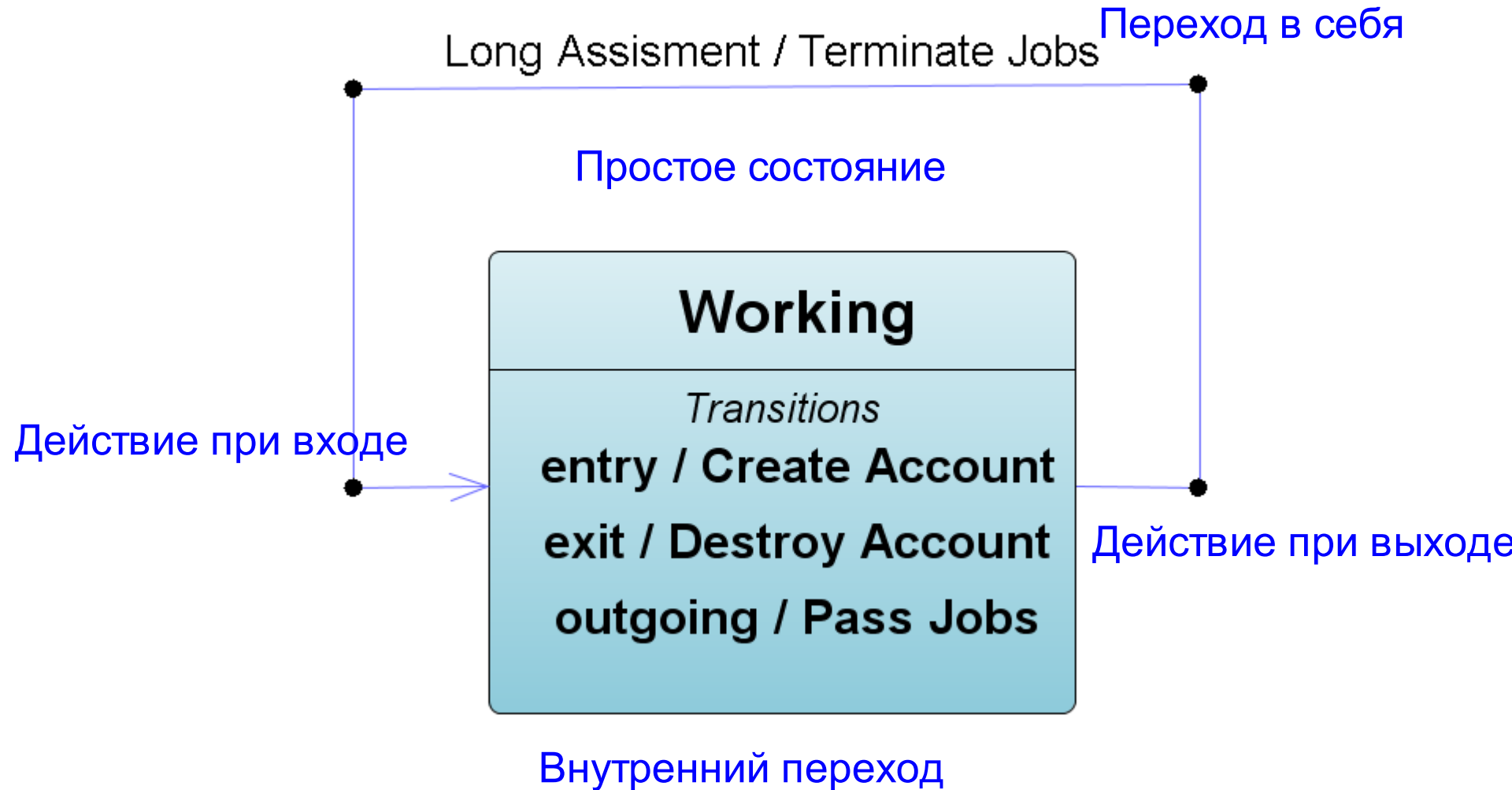
Метамодель машины состояний



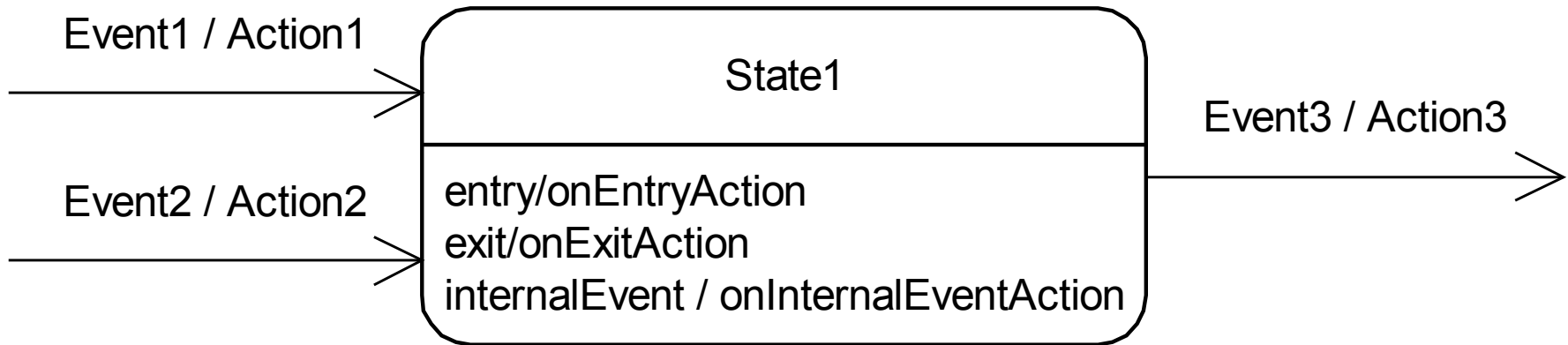
Простое состояние

- ИМЯ (в 2.0 м.б. анонимные состояния!)
- Действие при входе **entry**
- Действие при выходе **exit**
- Внутренние переходы (internal transitions)
 - ≠ переходы в себя
- Внутренняя активность **do**
 - по умолчанию - ожидание
- Отложенные события **defer**
 - Маскирование прерываний

Пример ИС ОК: простое состояние сотрудника

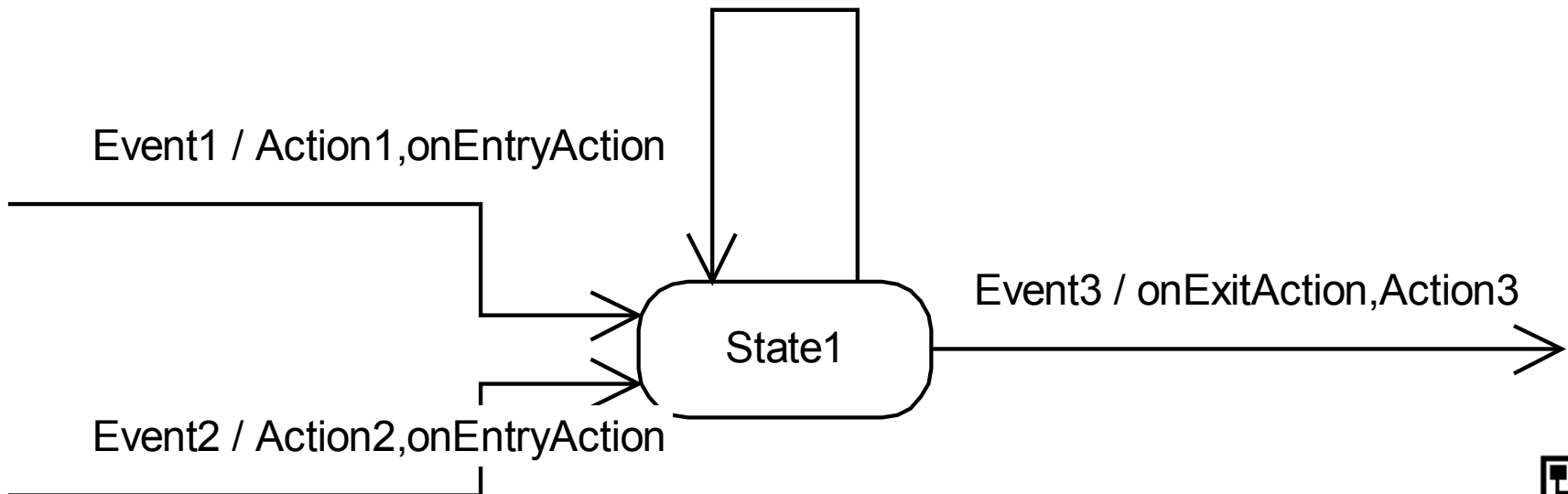


Исходный фрагмент

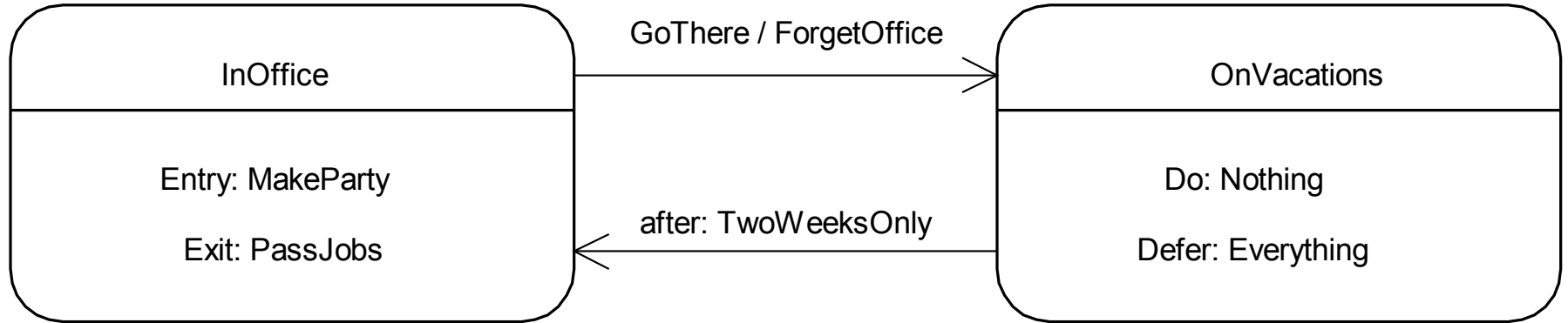


Эквивалентный фрагмент

internalEvent / onInternalEventAction



Отложенные события



Простой переход

- **СОБЫТИЕ**

- Сигнал, Вызов, Таймер, Изменение состояния
- Если возникает событие, переход возбуждается

- **[Сторожевое условие]**

- Если True, то переход срабатывает
- Если False, то переход не срабатывает и событие теряется

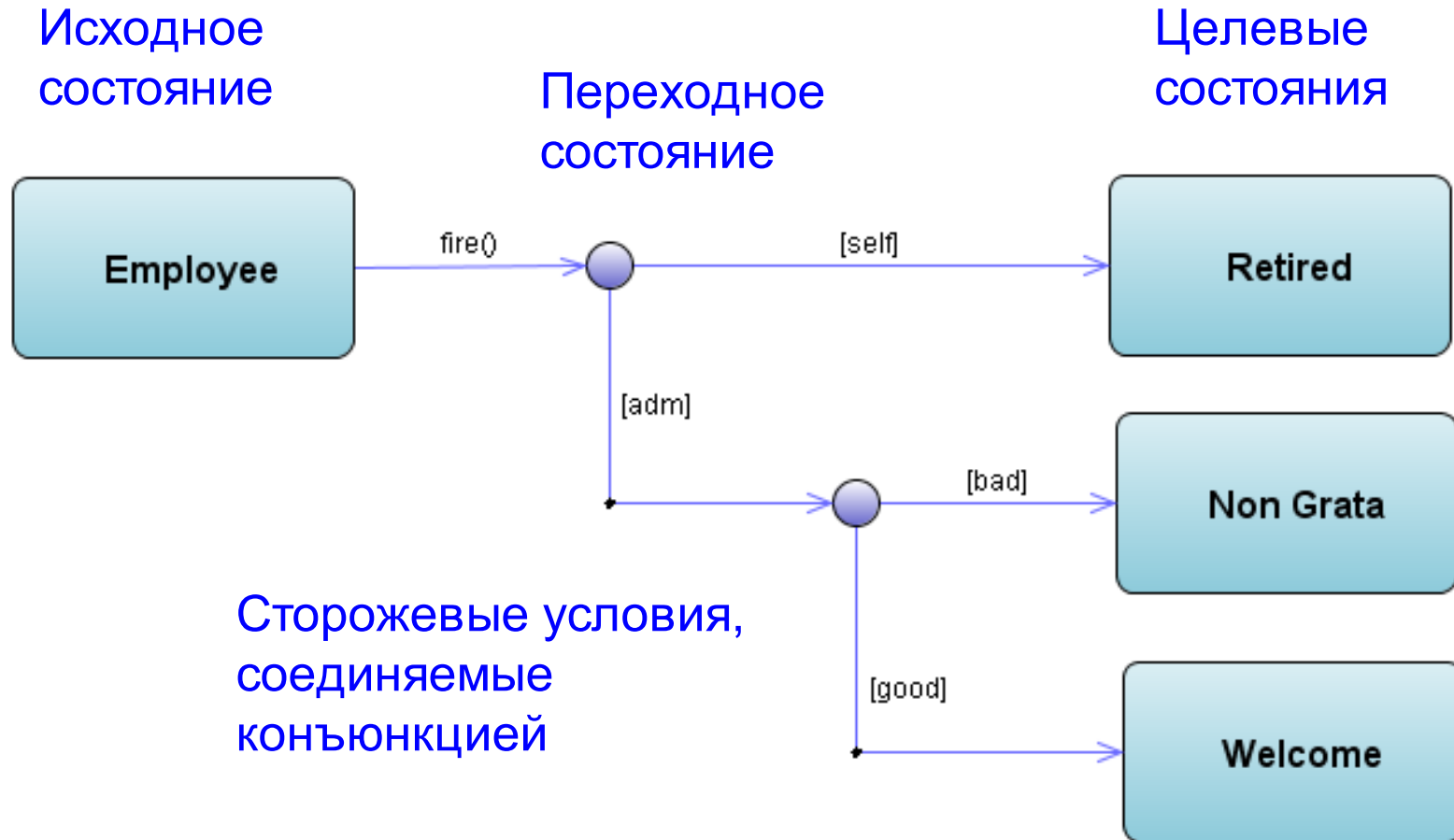
- **/ Действие**

- Выполняется, если переход срабатывает

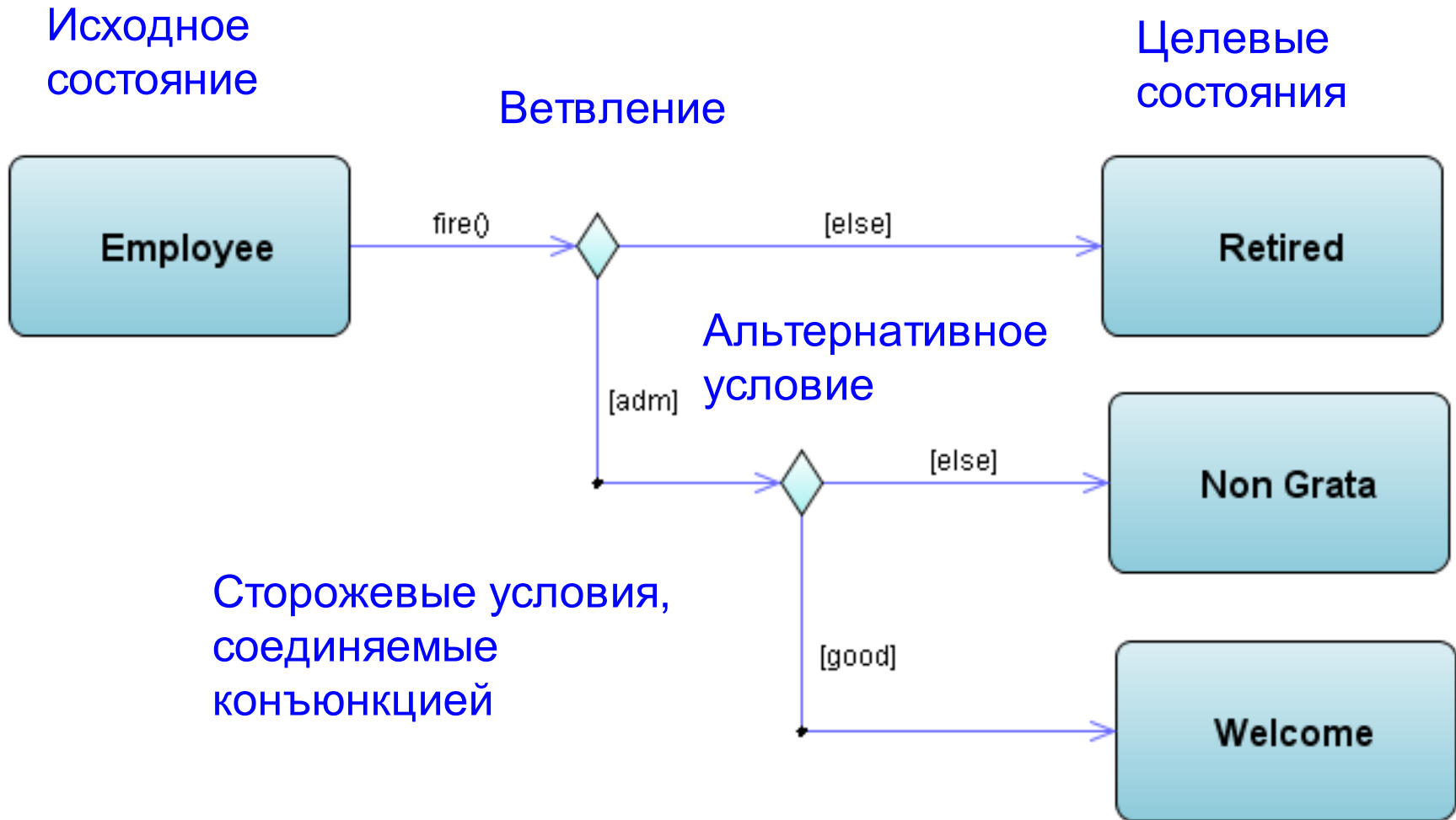
Пример ИС ОК: простой переход



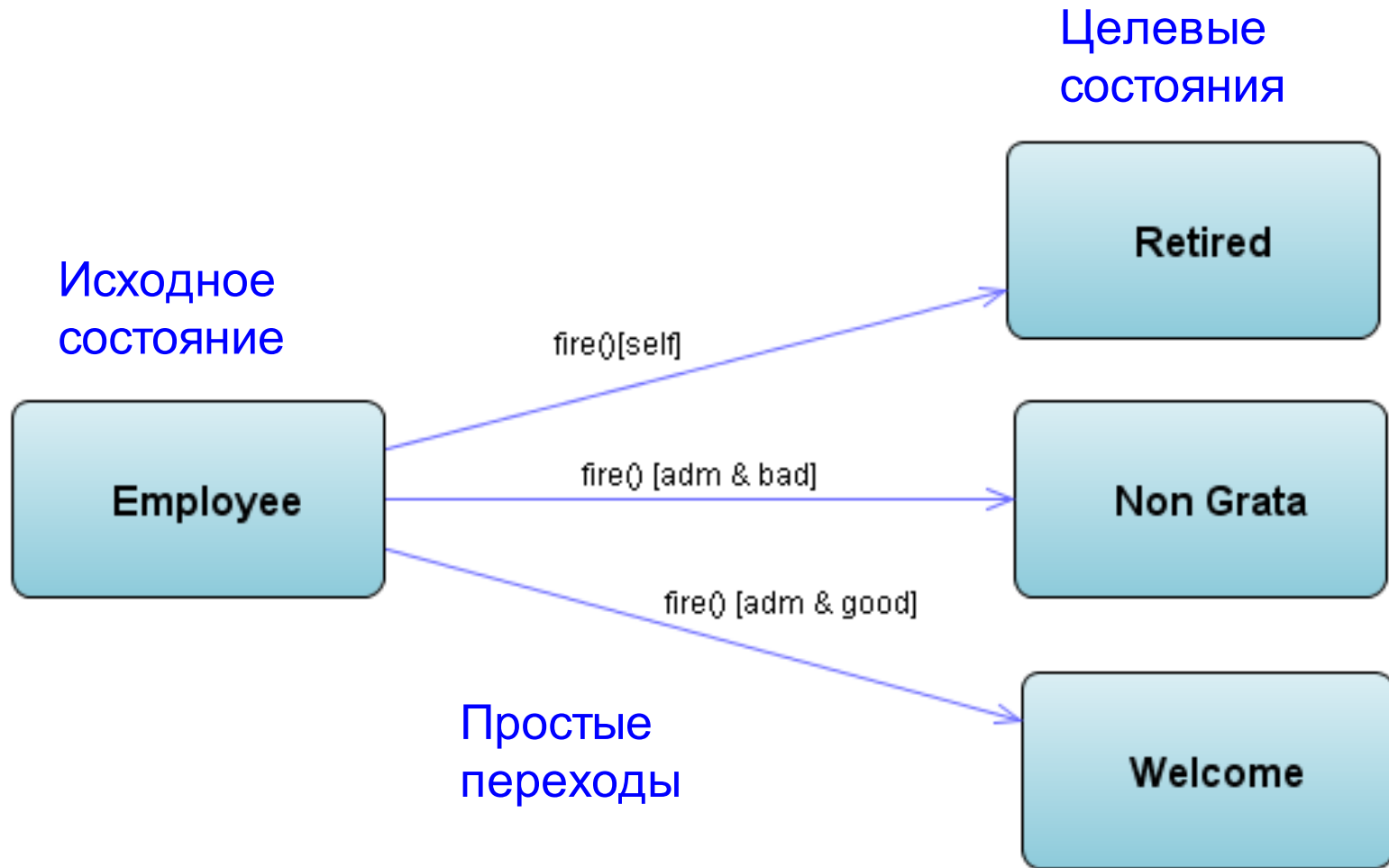
Сегментированные переходы и переходные состояния



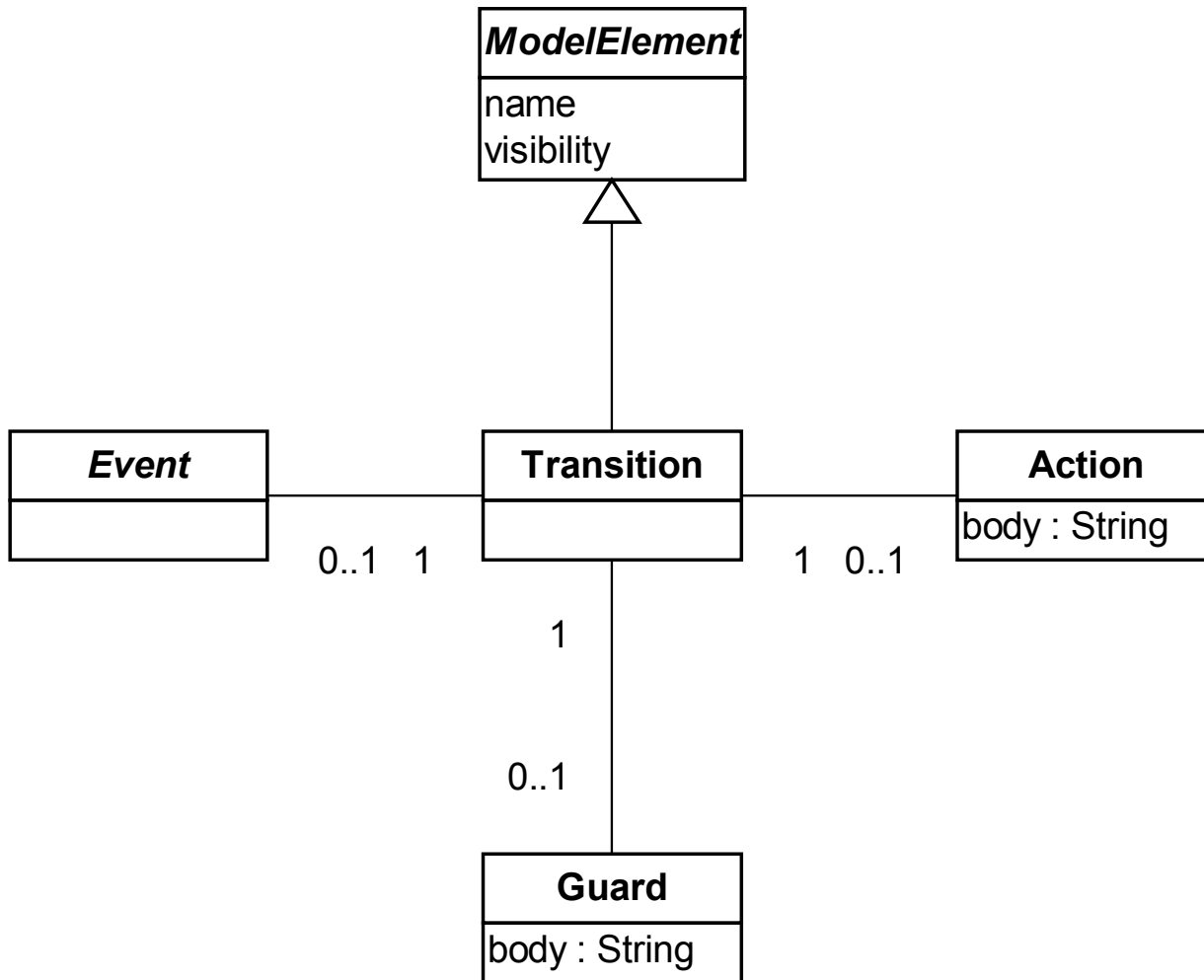
Использование предиката **else**



Использование простых переходов



Метамодель простого перехода

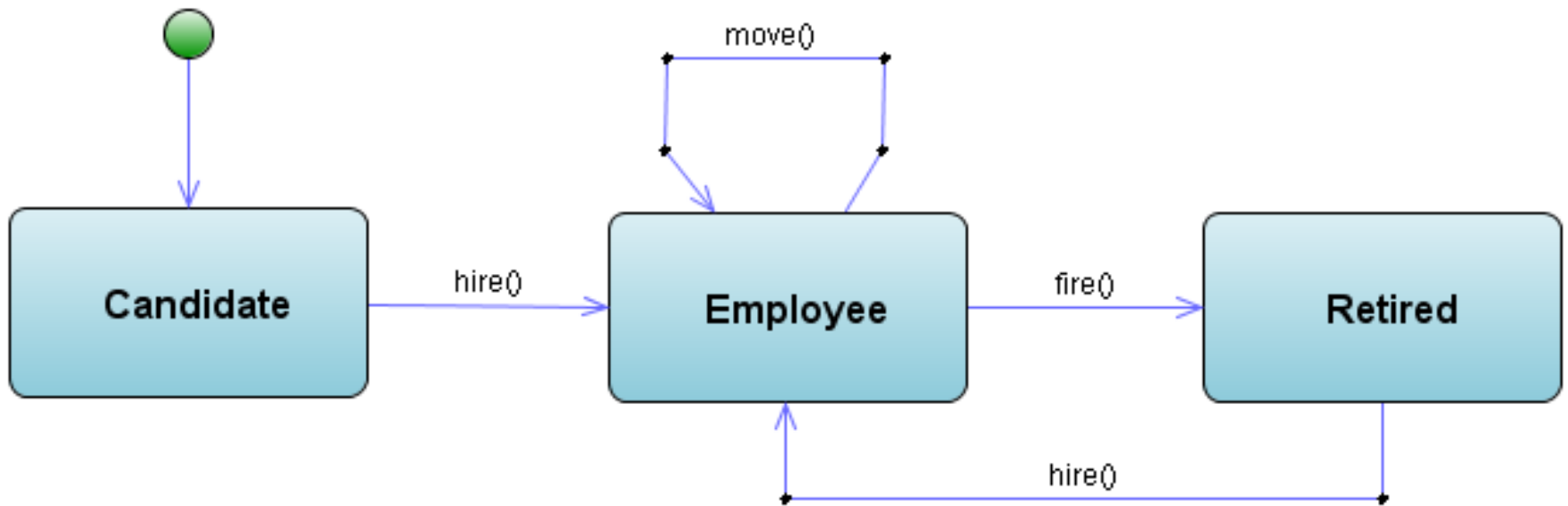


Пример: ИС ОК

таблица состояний сотрудника

	Прием	Перевод	Увольнение
Кандидат	Принять() Работник	Ошибка() Кандидат	Ошибка() Кандидат
Работник	Ошибка() Работник	Перевести() Работник	Уволить() Уволен
Уволен	Принять() Работник	Ошибка() Уволен	Ошибка() Уволен

Диаграмма состояний сотрудника



Составные и специальные состояния UML 1.x → 2.0

■ Составное состояние =

- Последовательное = вложенный автомат
- Параллельное (→ ортогональное) = несколько автоматов

■ Специальное состояние :

- начальное состояние
- заключительное состояние → в двух видах
- переходное состояние
- историческое состояние (в двух вариантах)
- синхронизирующее состояние ☞ в 2.0
- ссылочное состояние → вложенный автомат (submachine state)
- состояние "заглушка" → точки входа (entry point) и выхода (exit point)

Состояния сотрудника

Начальное
состояние

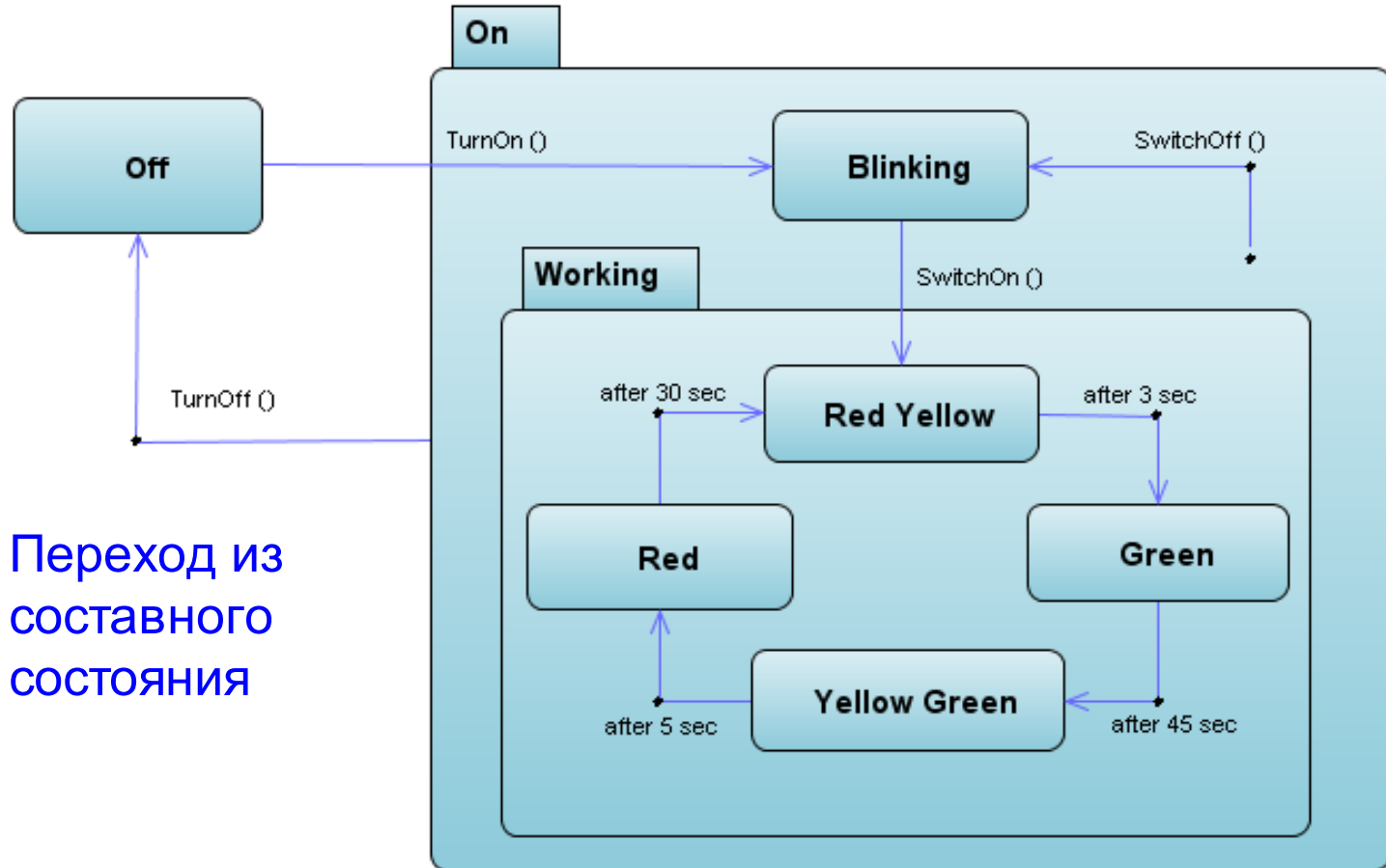


Простое
состояние



Светофор (составные состояния)

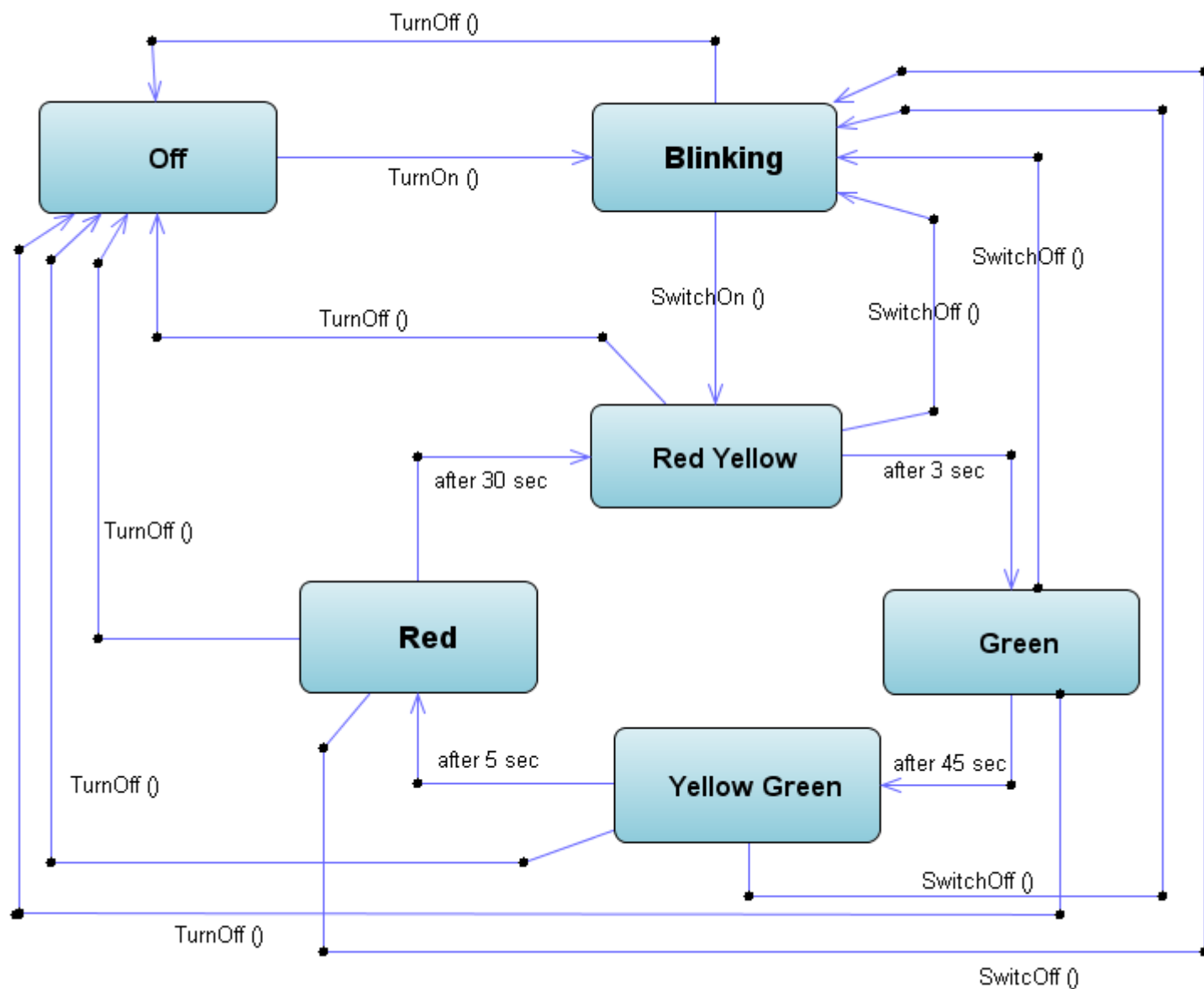
Составное состояние



Переход из
составного
состояния



Светофор (простые состояния)



Вредные советы 😊

- **Используйте имена состояний, которые предлагает инструмент: State1, State2 и т. д.**
- **Никогда не рисуйте более одного заключительного состояния.**
- **Нарисуйте столько вложенных состояний, сколько поместится на листе, но ни в коем случае не используйте переходы *между* составными состояниями.**
- **Не допускайте ситуации, когда переход из начального состояния ведет в простое состояние той же машины состояний: такой переход должен *пересекать* несколько границ составных состояний**

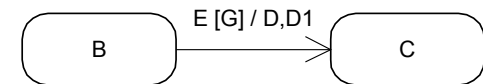
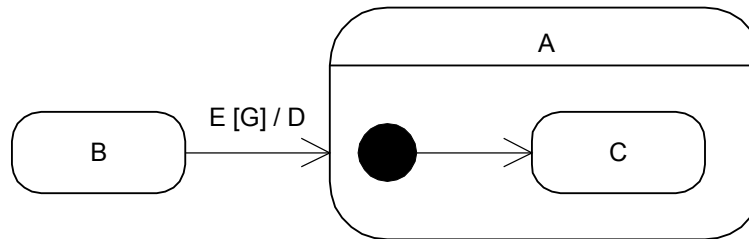
Переходы между составными состояниями

Вид перехода

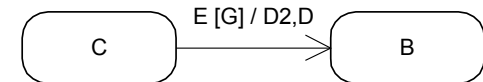
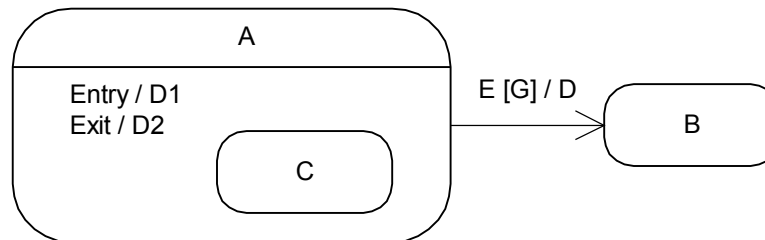
Диаграмма перехода

Эквивалентная диаграмма

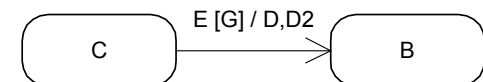
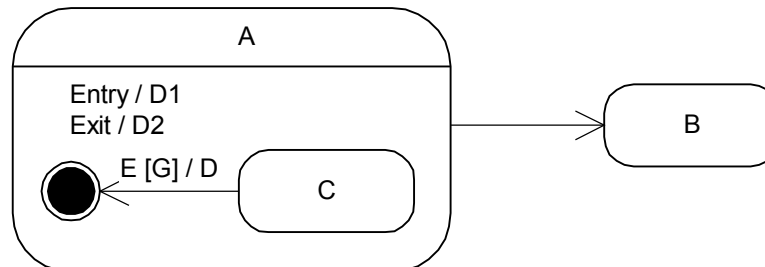
Переход в
составное
состояние



Переход из
составного
состояния

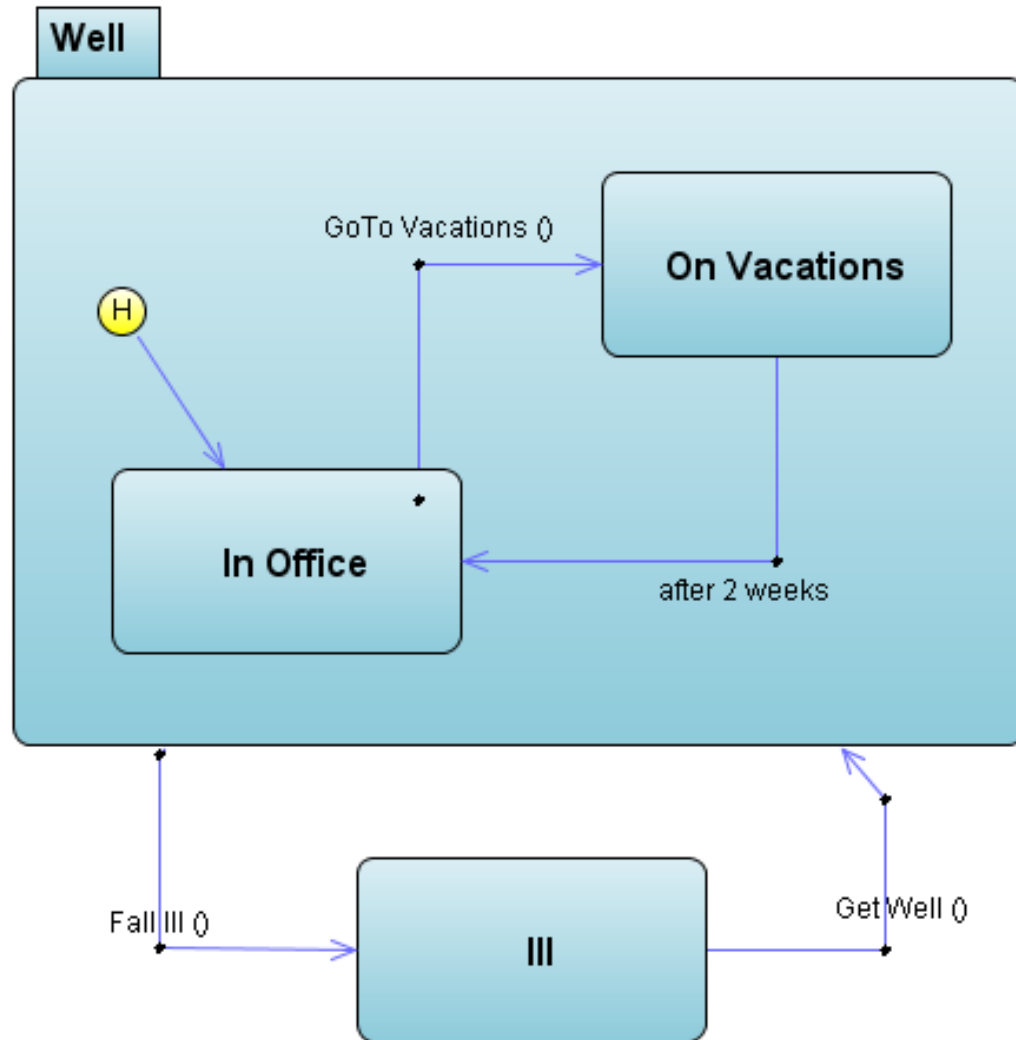


Переход по
завершении



Историческое состояние

Поверхностное
историческое
состояние

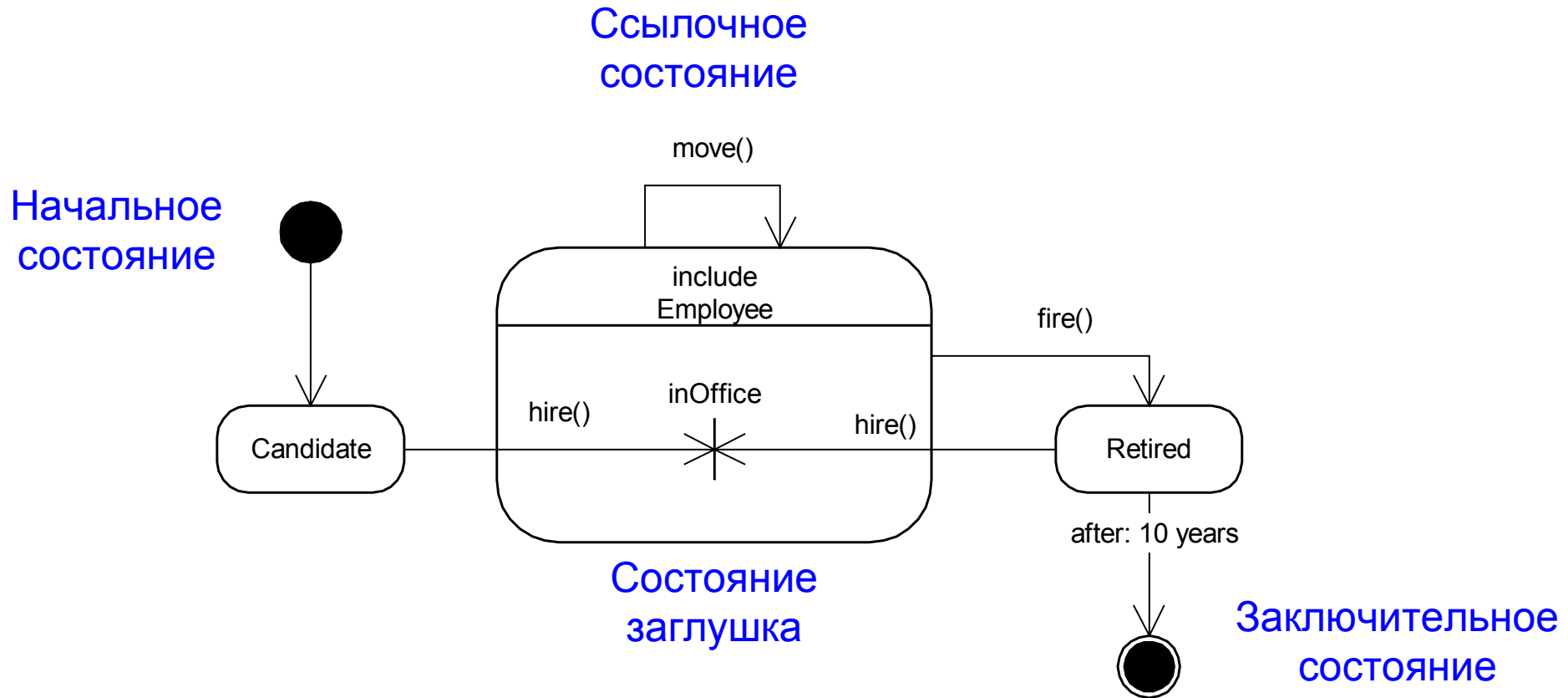


Составное
состояние с
вложенным
историческим
состоянием

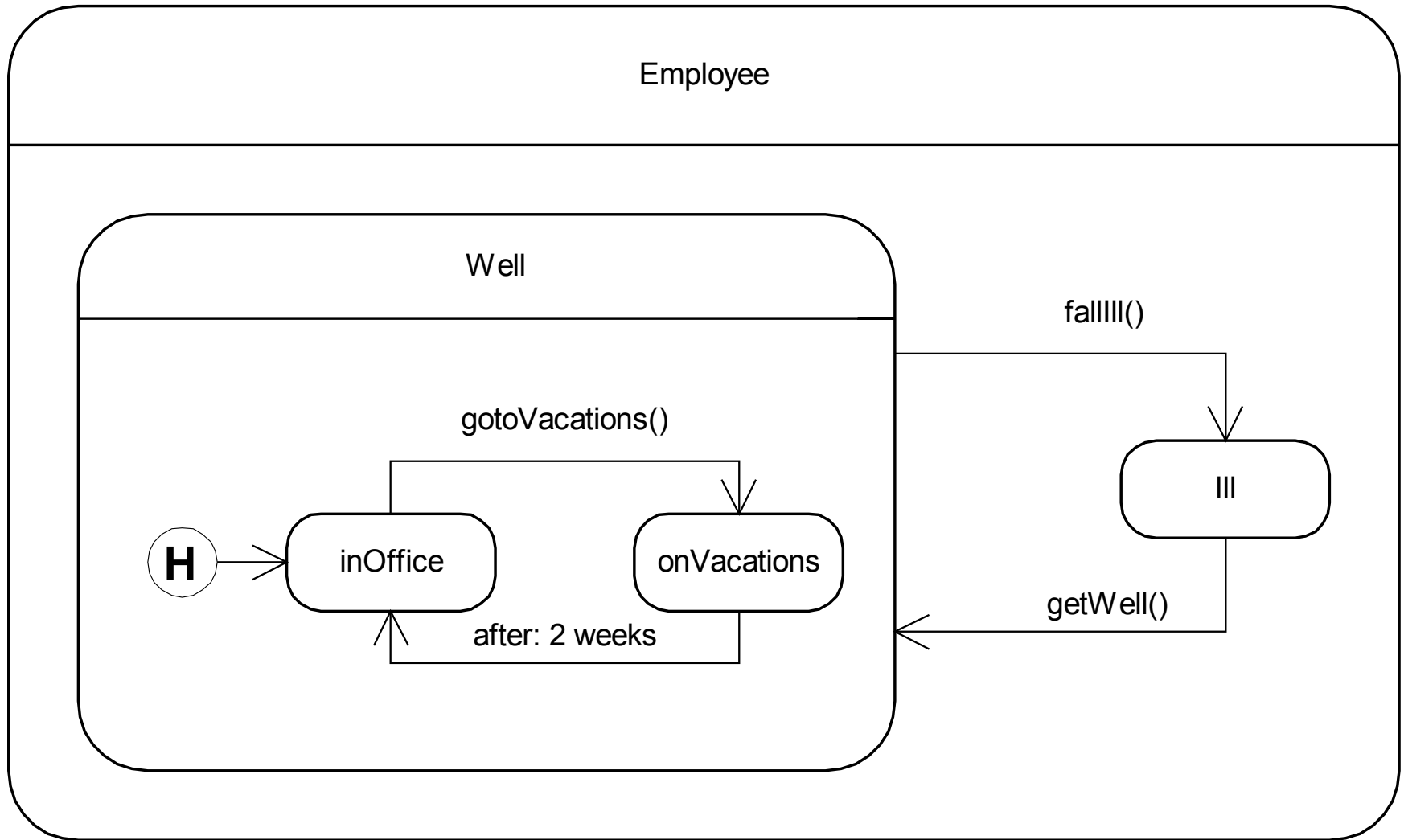
Простое
состояние



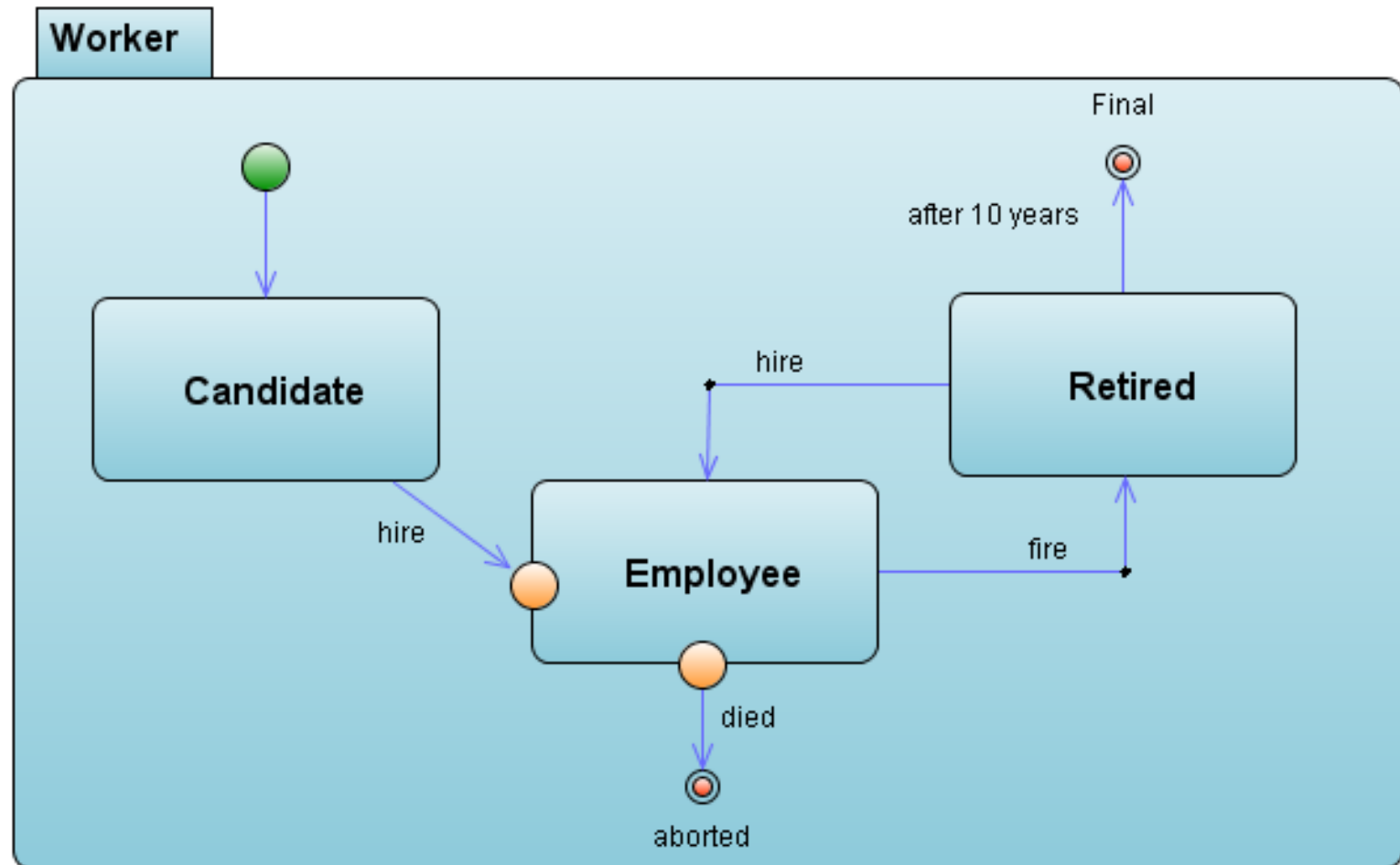
Ссылочное состояние и заглушка (1.4)



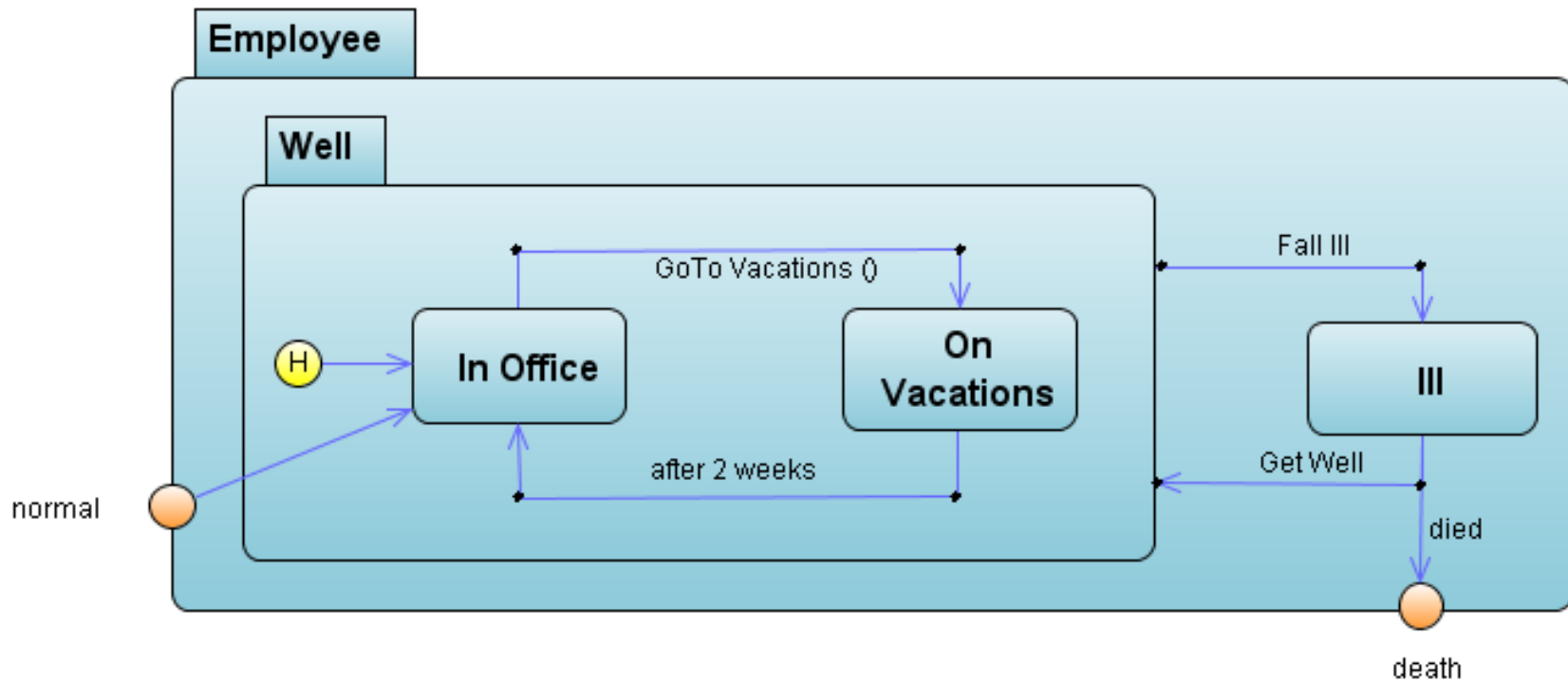
Ссылочное состояние и заглушка (1.4)



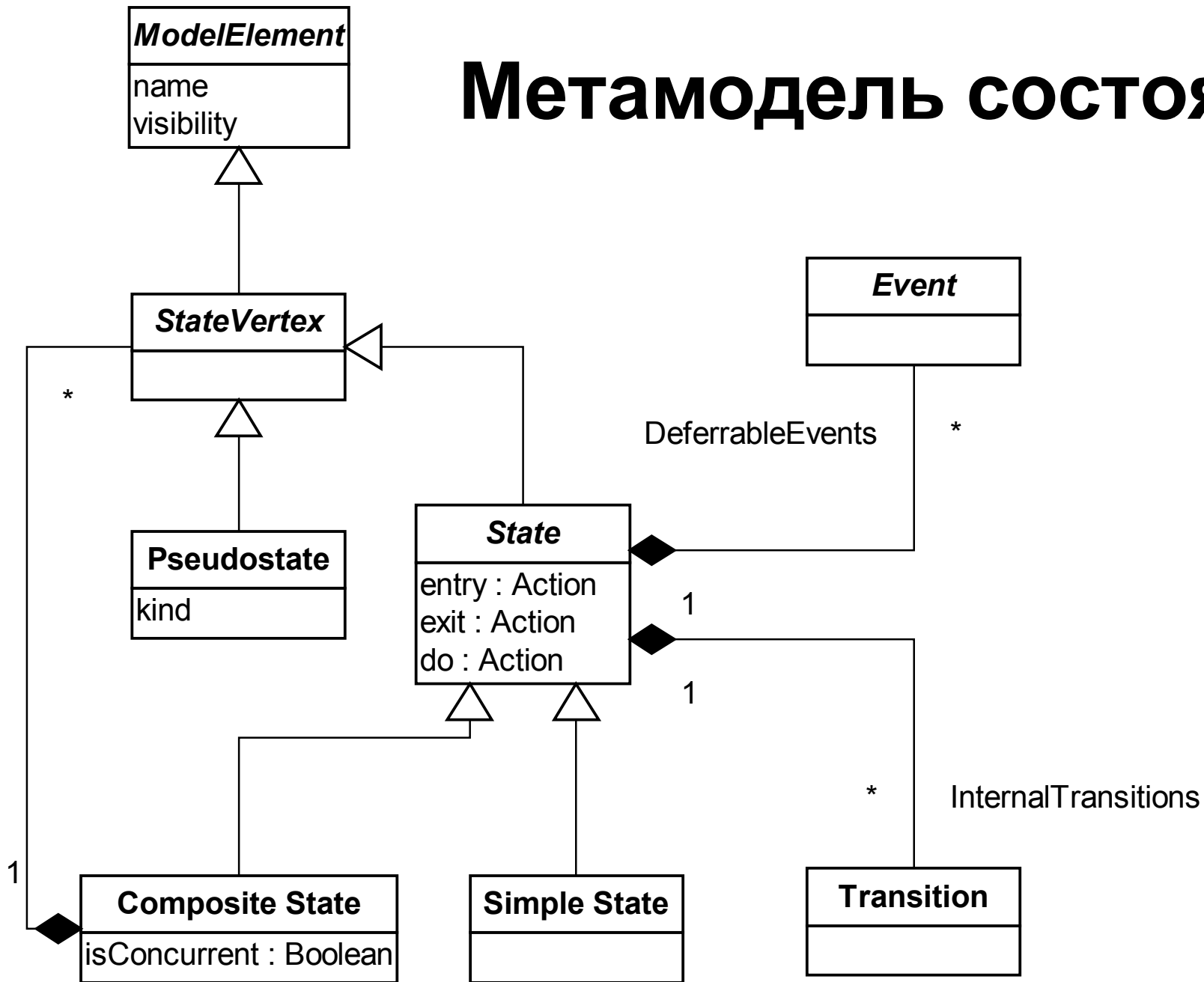
Вложенные машины состояний (1)



Вложенные машины состояний (2)



Метамодель состояния

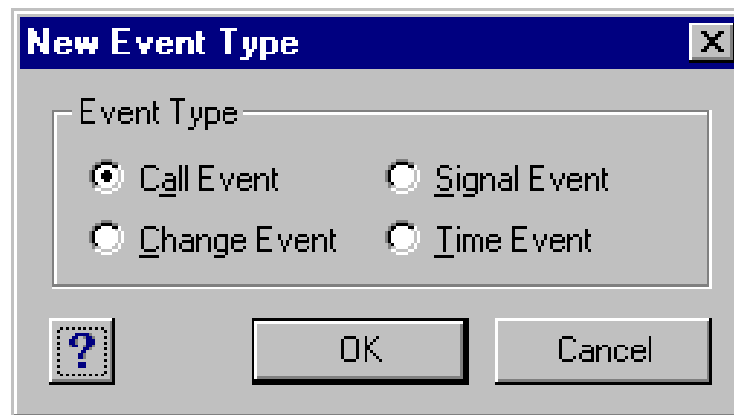


События

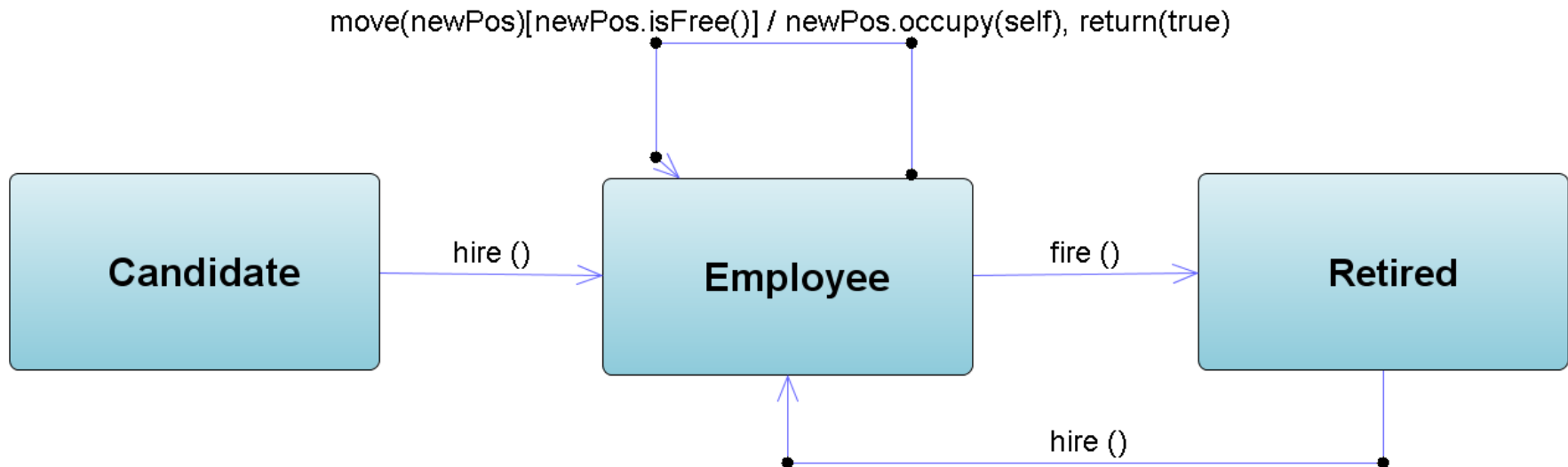
- **События:**
 - Внешние – передаются между системой и действующими лицами
 - Внутренние – передаются между объектами внутри системы
 - В UML 1.4 не различаются, в UML 2.0 различаются – событие от точки входа
- **Типы событий UML:**
 - Вызов (Call event)
 - Изменение (Change event)
 - Таймер (Time event)
 - Сигнал (Signal event)

Событие вызова

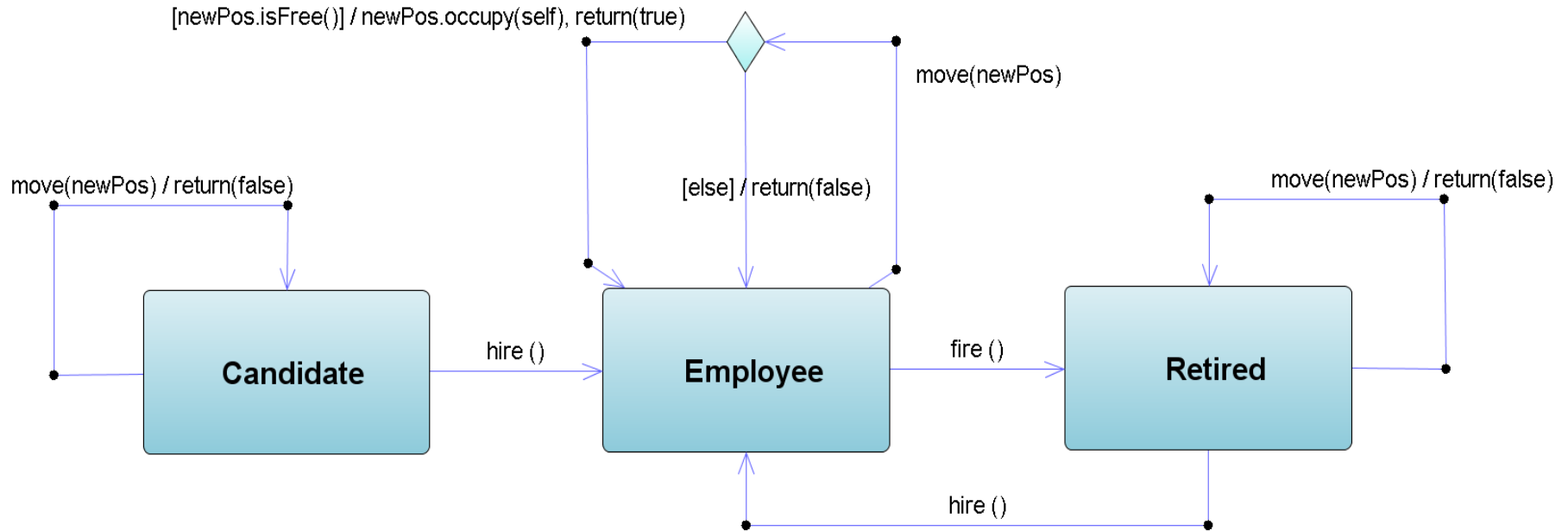
- Автомат моделирует поведения объекта – вызов операции объекта
- Может иметь аргументы и возвращать результат
- Самый распространенный случай



Пример ИС ОК: move (1)

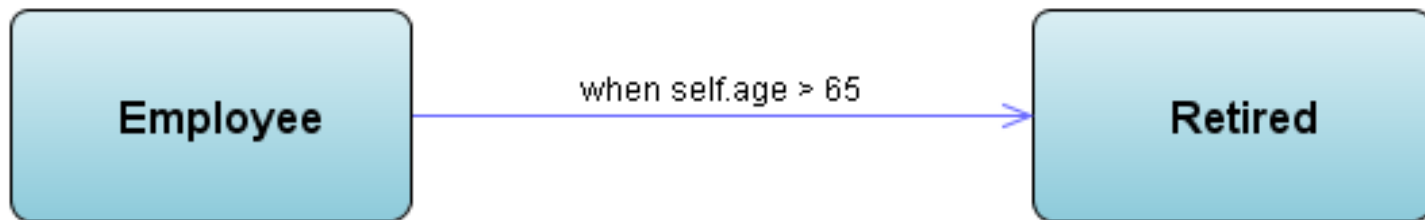


Пример ИС ОК: move (2)



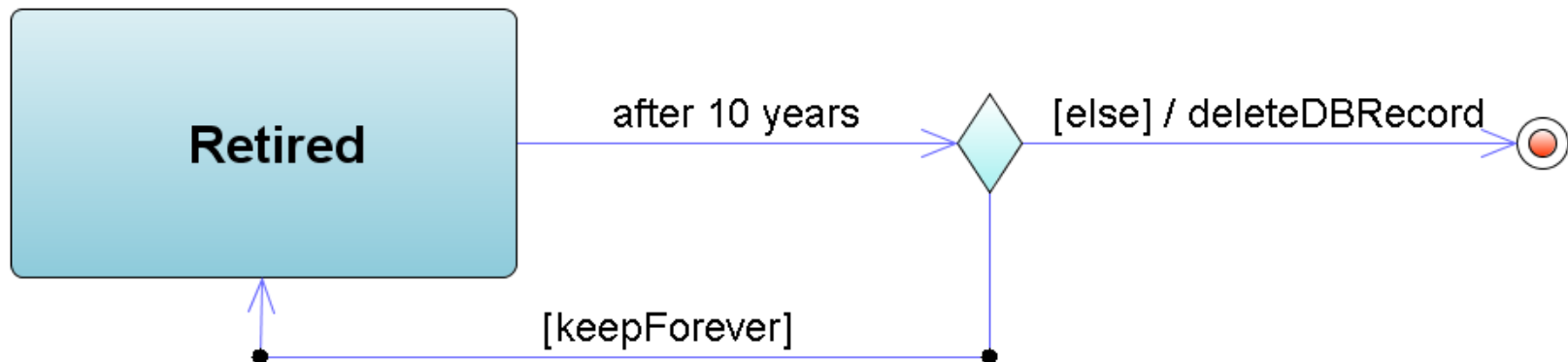
Событие изменения

- **When** (логическое выражение)
- **Событие изменения наступает, когда значение выражения становится true**
- **Возможен эффективный демон:**
 - **Прямые указатели от использующих вхождений к определяющему**
 - **Обратные указатели от определяющего вхождения к использующим**



Событие таймера

- **After**(выражение, доставляющее длину интервала времени)
- События времени применяются на переходах в автоматах
- Отсчет времени (момент 0) связывается с моментом входа в текущее состояние
- Таймеры состояний независимы



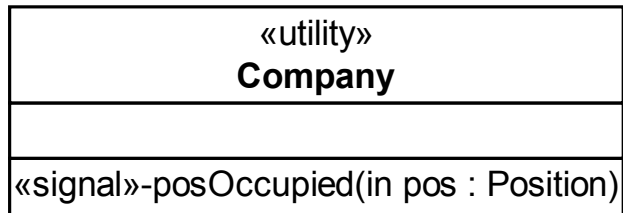
Таймер и «абсолютное» время

- В стандарте time event, но подразумеваются ЧАСЫ, а не ВРЕМЯ
- Перевод: «событие таймера», а не «событие во времени»
- Чтобы задать переход по астрономическому времени (системным часам!), используется СОБЫТИЕ ИЗМЕНЕНИЯ:
- When (date = “1 апреля 2006г”)

Событие сигнала

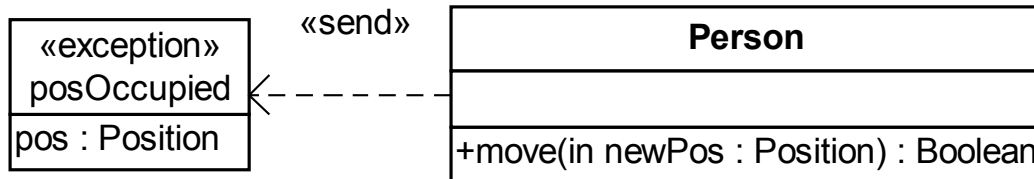
- Событие сигнала наступает, когда послан сигнал
- Сигнал – класс со стереотипом «signal»
- Атрибуты сигнала = параметры
- Возможная иерархия обобщений сигналов
- Посылка сигнала = создание объекта
- Класс принимает сигнал = имеет операцию с именем сигнала и стереотипом «signal»
- Широковещательный сигнал: посылается множеству объектов
- Исключение = частный случай сигнала

Описание сигнала



Класс,
обрабатывающий
исключение

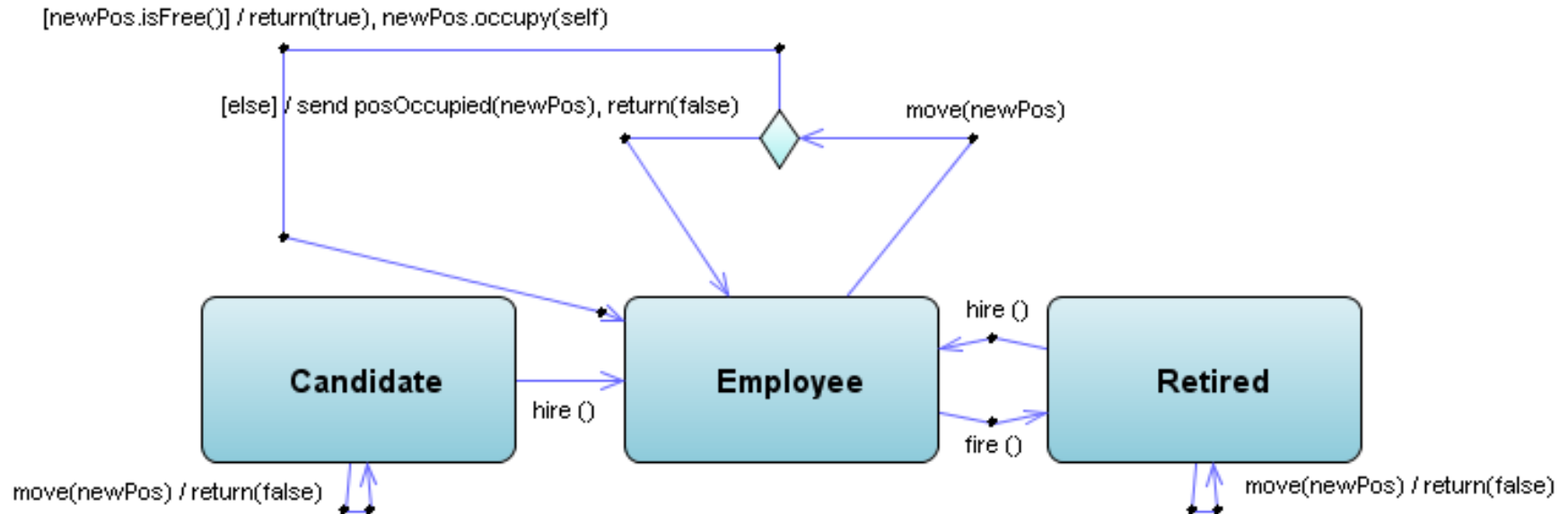
Исключение



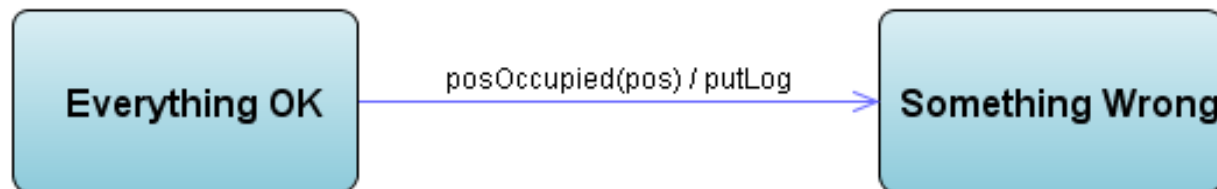
Класс,
возбуждающий
исключение



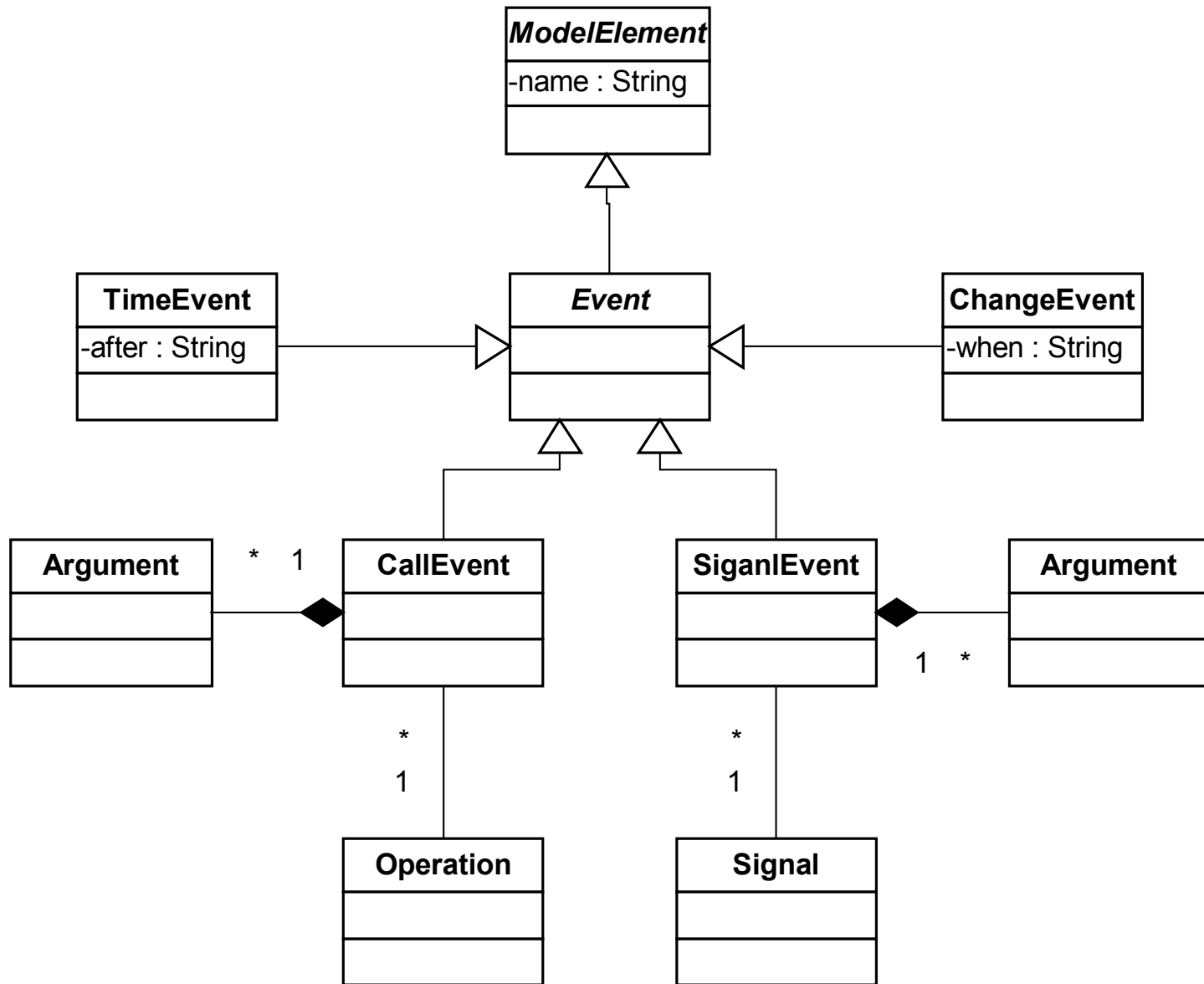
Посылка сигнала



Переход по событию сигнала

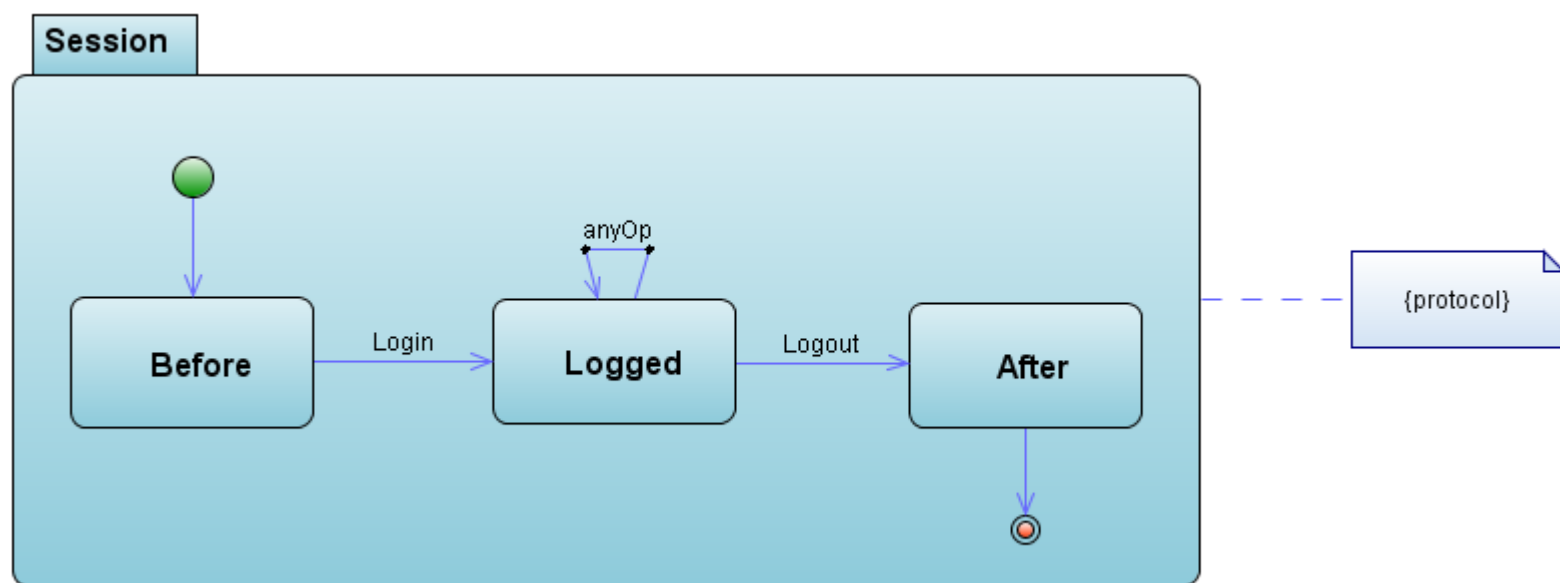


Метамодель события



Протокольный автомат

- Предназначен для задания допустимых последовательностей операций
- На переходах нет действий
- Ключевое слово {protocol}



5.3. Диаграммы деятельности

- Что нового в UML 2.0
- Деятельность (activity)
- Действие (action)
- Переходы (transitions)
- Ветвления (decisions)
- Развилки и слияния (fork & join)
- Дорожки (swim lanes)
- Траектория объекта (object flow)
- Посылка и прием сигналов
- Прерывания и исключения

Что нового в UML 2.0

- Независимая семантика маркеров
- Потоки управления и/или потоки данных
- Вложенность (структурная декомпозиция)
- Семантика разделов (дорожек)
- Обработка множеств объектов
- Параметризация деятельности
- Расширенный набор действий (из 1.5)
- Исключения и прерывания
- Предусловия и постусловия действий

Семантика маркеров в UML 2.0

- **Маркеры (tokens)**
 - Маркер управления (control flow token) (пустой)
 - Маркер данных (data flow token) (со ссылкой на объект или структуру данных)
- **Узлы – действия с контактами (pins), параметры, объекты (data store), ...**
- **Ребра управления и ребра данных соединяют узлы**
- **Маркеры возникают, перемещаются и исчезают**
 - Начальное состояние создает маркер управления
 - Параметр деятельности создает маркер данных
 - Входной контакт действия поглощает маркер данных
 - Выходной контакт действия создает маркер данных
 - Заключительное состояние поглощает маркер управления
 - Развилка размножает маркер управления
 - Соединение склеивает маркеры управления
 - Ветвление направляет маркер управления или данных
- **Условие выполнения действия**
 - Ребра данных, входящие во все входные контакты, готовы передать маркеры данных
 - Если есть входящие ребра управления, то хотя бы одно готово передать маркер управления
- **Результат выполнения действия**
 - Все ребра данных, выходящие из выходных контактов, готовы передать маркеры данных
 - Если есть исходящие ребра управления, то все готовы передать маркер управления (но маркер один – конкуренция, кто первый готов, тот и получит)

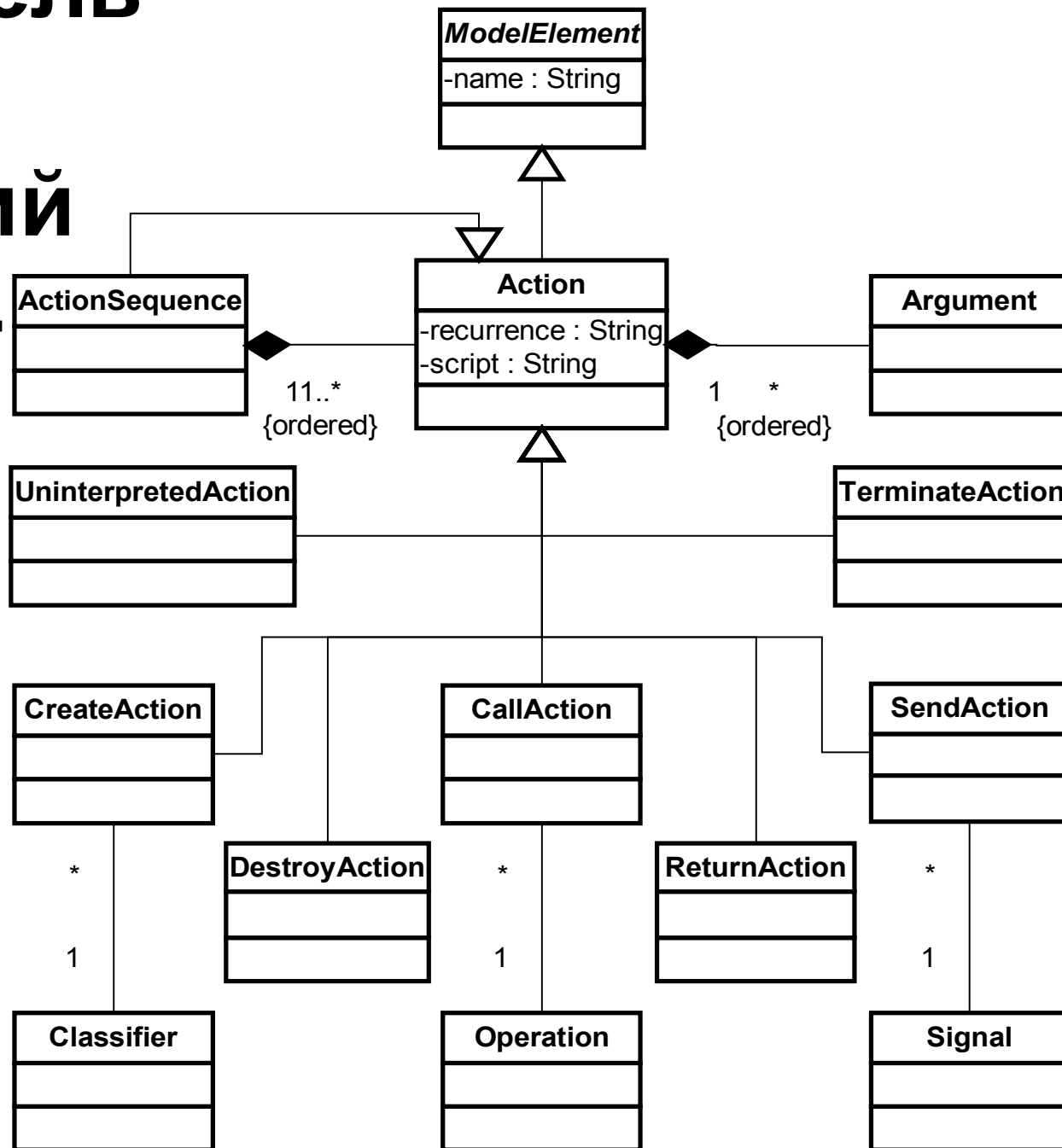
Действие и деятельность в UML 1.x

- **Действие (Action)**
 - Атомарно (не может быть прервано)
 - Завершается (всегда и само)
 - Мгновенно (время выполнения мало)
 - Последовательность действий = действие
 - * [повторитель] действие
- **Деятельность (Activity)**
 - Может быть прервана событием
 - Может продолжаться неограниченно долго

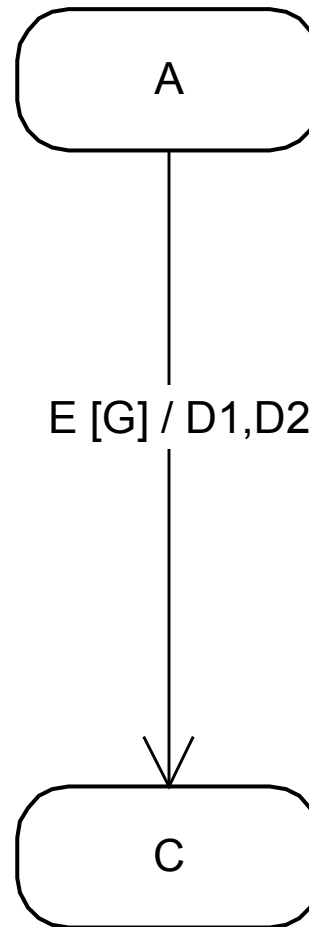
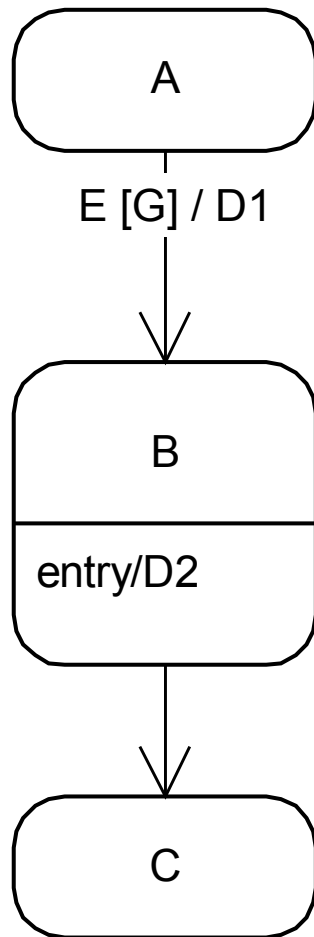
Типы действий в UML 1.1-1.4

- присваивание значения :=
- вызов операции call
- создание объекта new
- уничтожение объекта destroy
- возврат значения return
- посылка сигнала send
- останов terminate
- не интерпретируемое действие

действий 1.1-1.4



Состояние действия в UML 1.x – лишнее



Виды действий в UML 1.5-2.0 (из 53)

Категория	Действие	Описание
Классификация	readIsClassifiedObject	Проверка класса
	reclassifyObject	Изменение класса
	readExtent	Получение всех объектов класса
Коммуникация	broadcastSignal	Широковещательная передача сигнала
	sendObject	Отправка сигнала как объекта
	sendSignal	Отправка сигнала со списком аргументов
Вычисление	applyFunction	Вычисление без побочных эффектов
	opaque	Определяется реализацией

Виды действий в UML 1.5-2.0 (из 53)

Категория	Действие	Описание
Управление	startClassifierBehavior	Запуск потока управления
	callOperation	Обычный вызов метода
	callBehavior	Вызов процедуры
	raiseException	Возбуждение исключения
Чтение и запись	readStructuralFeature	Получение значения атрибута
	readLinkObjectEnd	Получение значения по связи
	writeStructuralFeature	Присваивание значения атрибуту
Создание и уничтожение	createObject	Создание объекта
	destroyObject	Удаление объекта

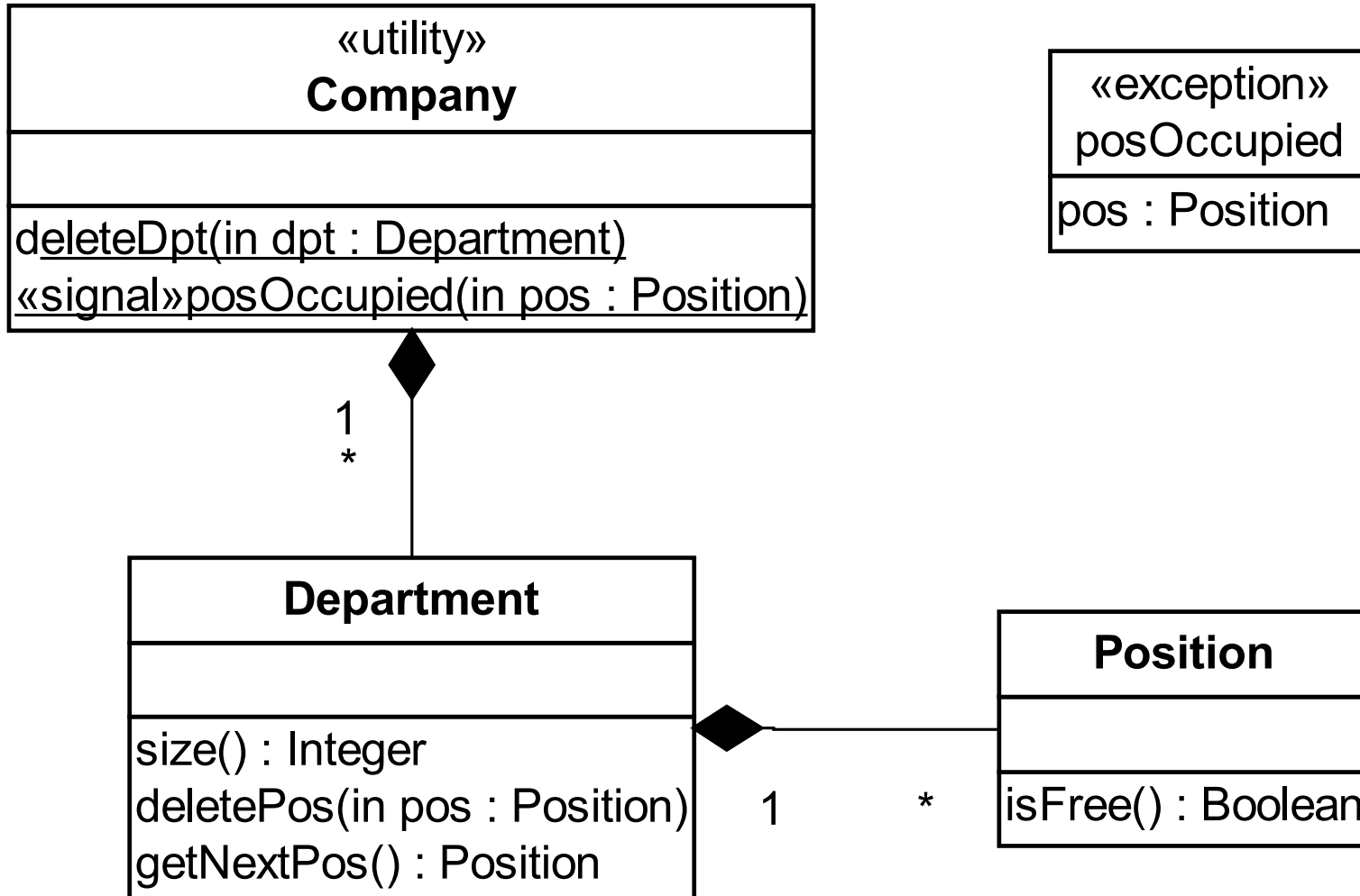
Элементы диаграммы деятельности 1.x→2.0

- **Состояние деятельности → действие**
 - 1.x: простое состояние и внутренняя активность (entry) является деятельностью
 - 2.0: действие, в т.ч. вызов деятельности
- **Выражение деятельности**
 - 1.x: синтаксис не определен
 - 2.0: синтаксис действий определяет инструмент
- **Переход**
 - По завершении (можно и по событию!)
 - По передаче данных
 - Сегментированный (ветвления и параллелизм)
 - Со сторожевыми условиями
 - С действиями на переходе

Связь диаграмм деятельности и состояний

- Пример ИС ОК: реализация операции **Delete Department**
- Используется семантика UML 1.x: диаграмма деятельности – частный случай машины состояний
- Постепенная трансформация блок-схемы в граф переходов
- Улучшение кода на уровне диаграмм

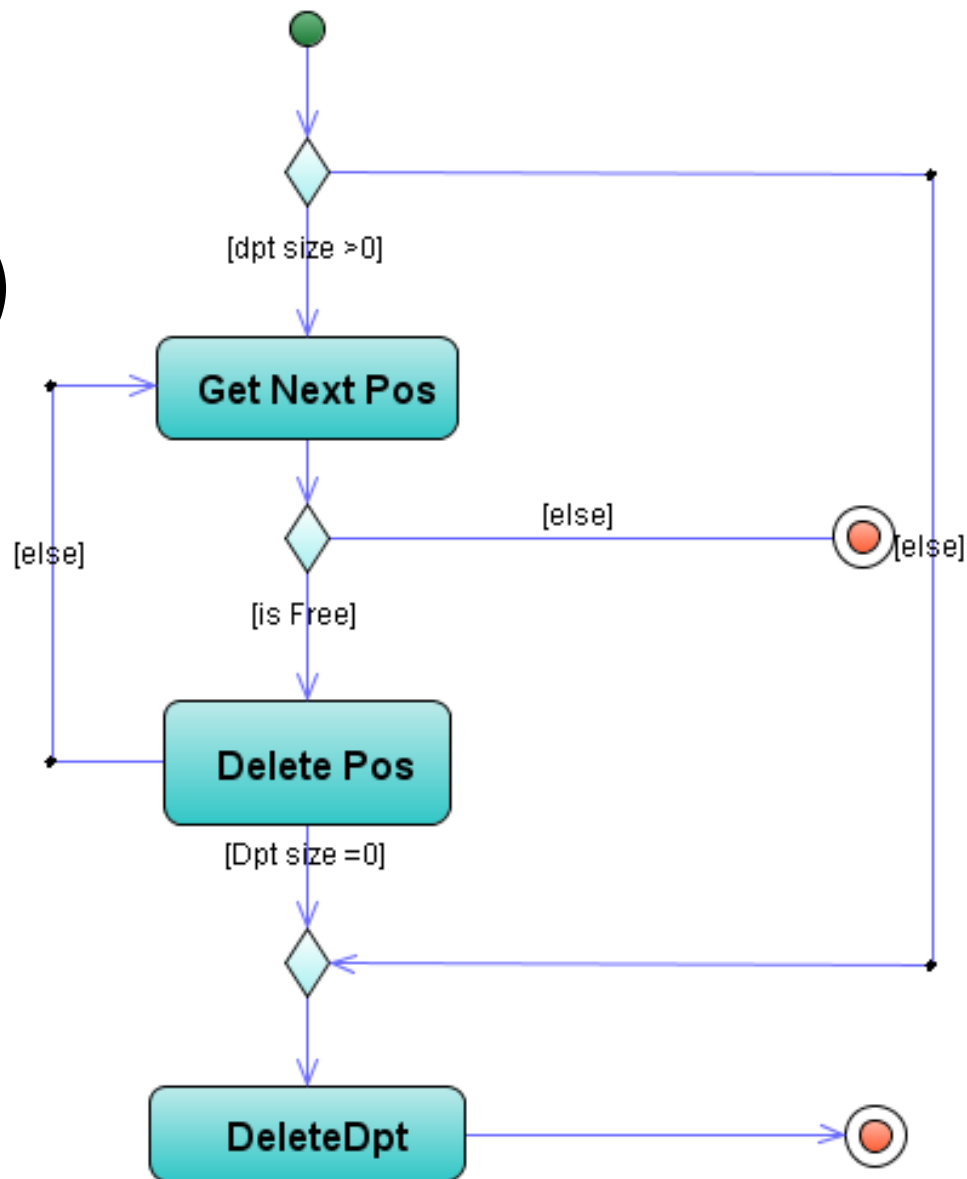
Delete Department: классы



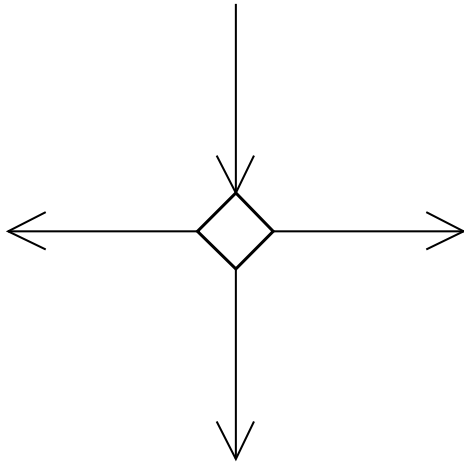
Delete Department: Код (1)

- **if dpt.size() > 0**
- **repeat**
- **pos := dpt.getNextPos()**
- **if not pos.isFree() then**
- **send posOccupied(pos)**
- **stop**
- **dpt.deletePos(pos)**
- **until dpt.size() = 0**
- **deleteDpt(dpt)**

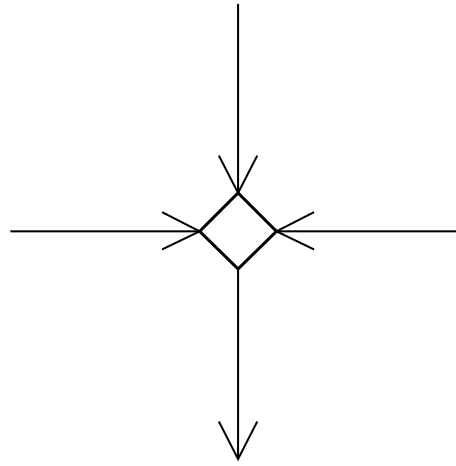
Delete Department: диаграмма (1)



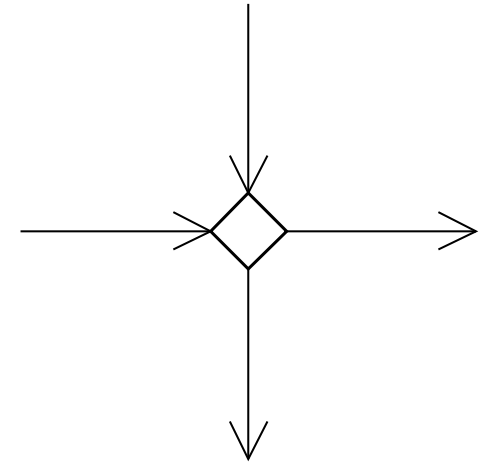
Что бы это значило? (в UML 1.x)



Ветвление



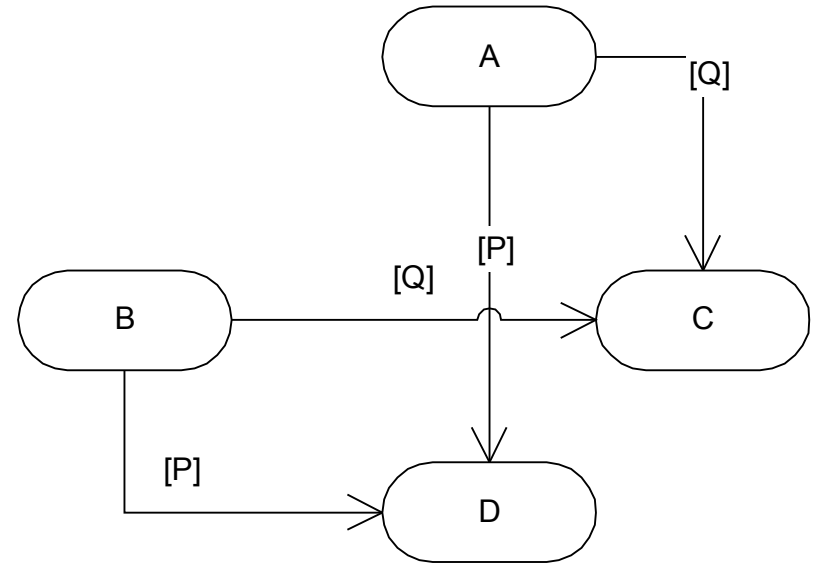
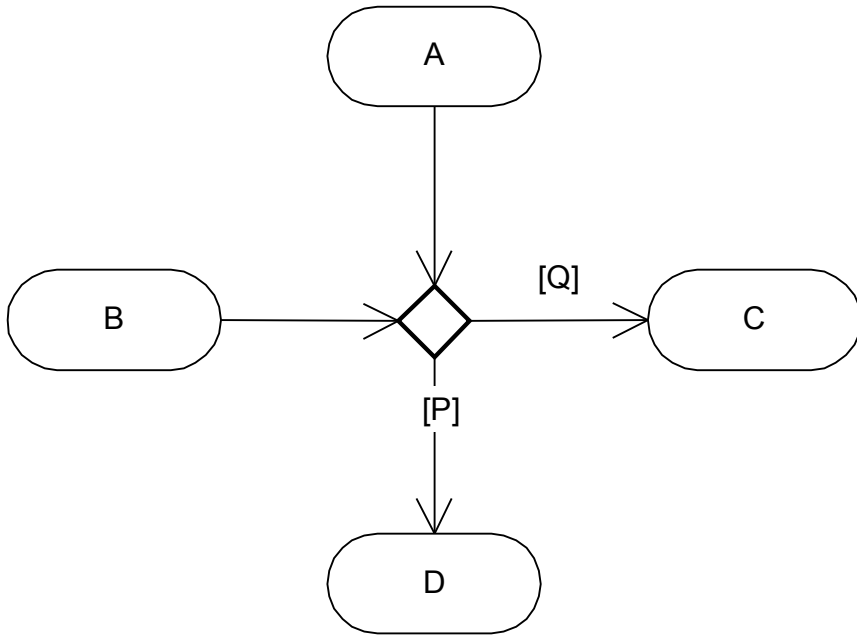
Слияние



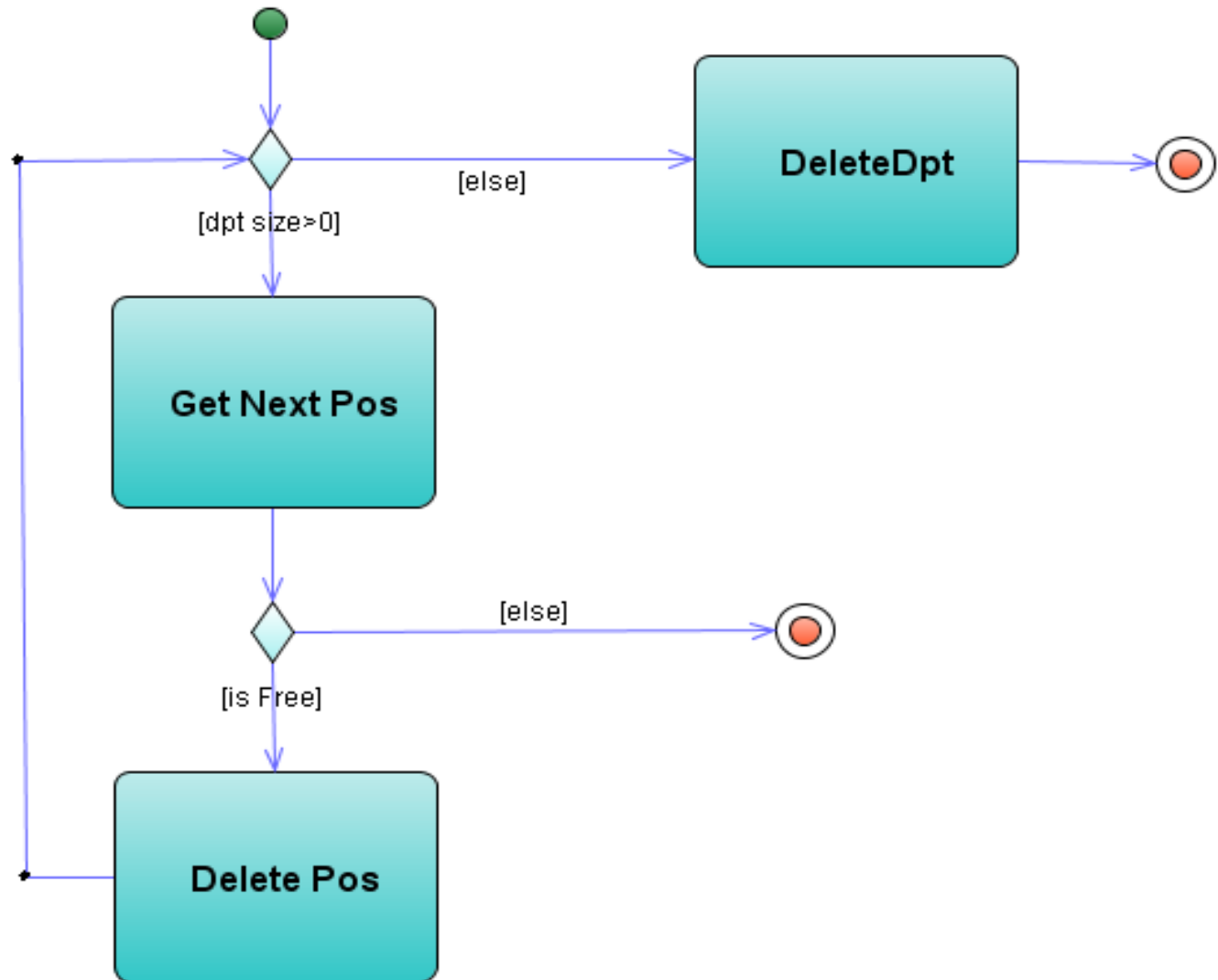
???



Семантика «? ? ?» (в UML 2.0)



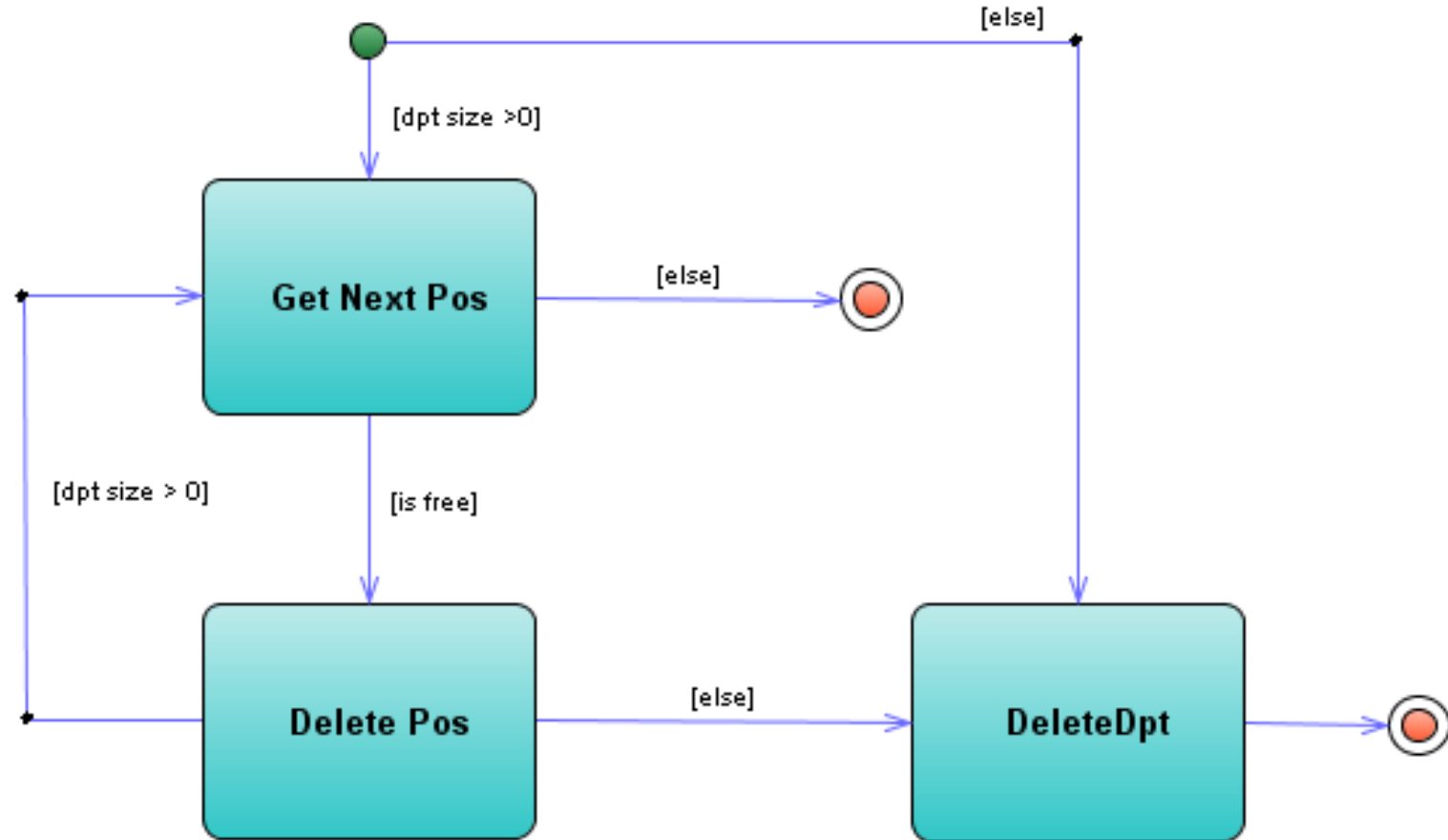
Delete Department: диаграмма (2)



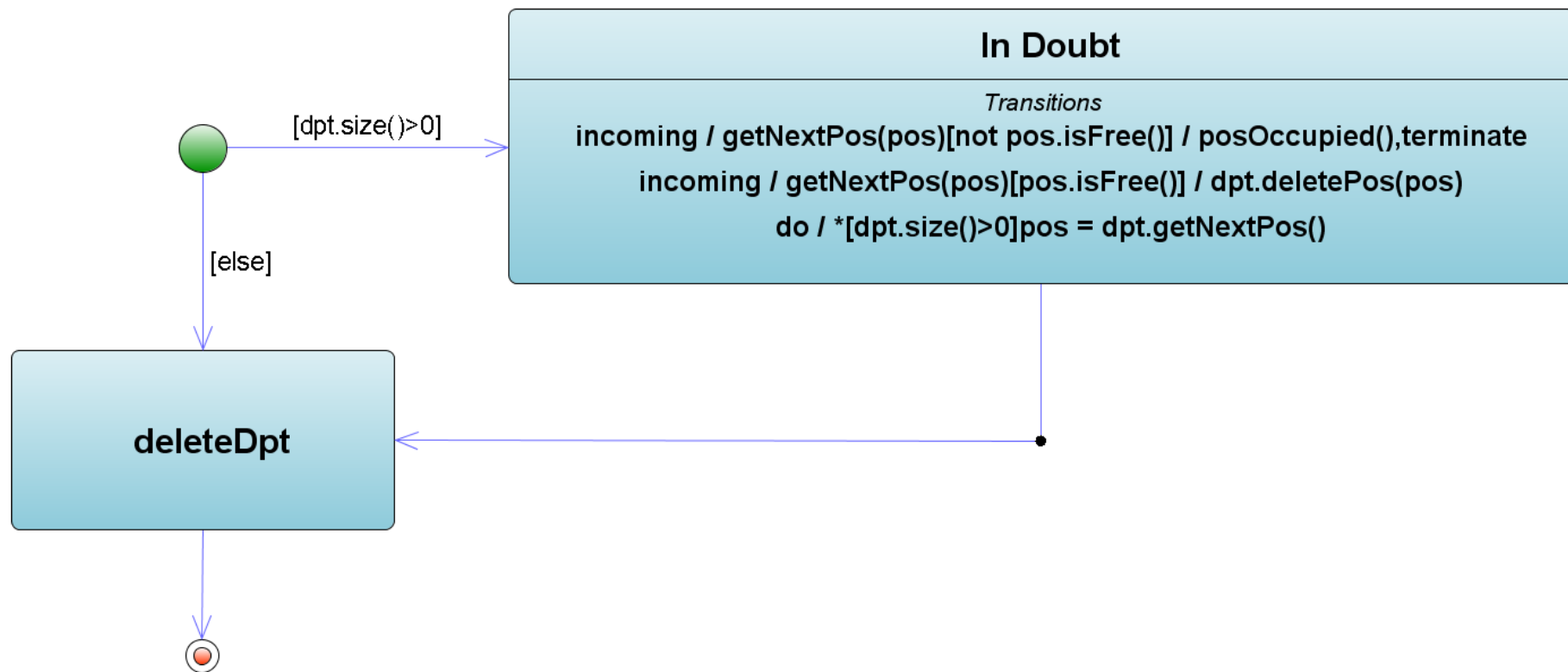
Delete Department: код (2)

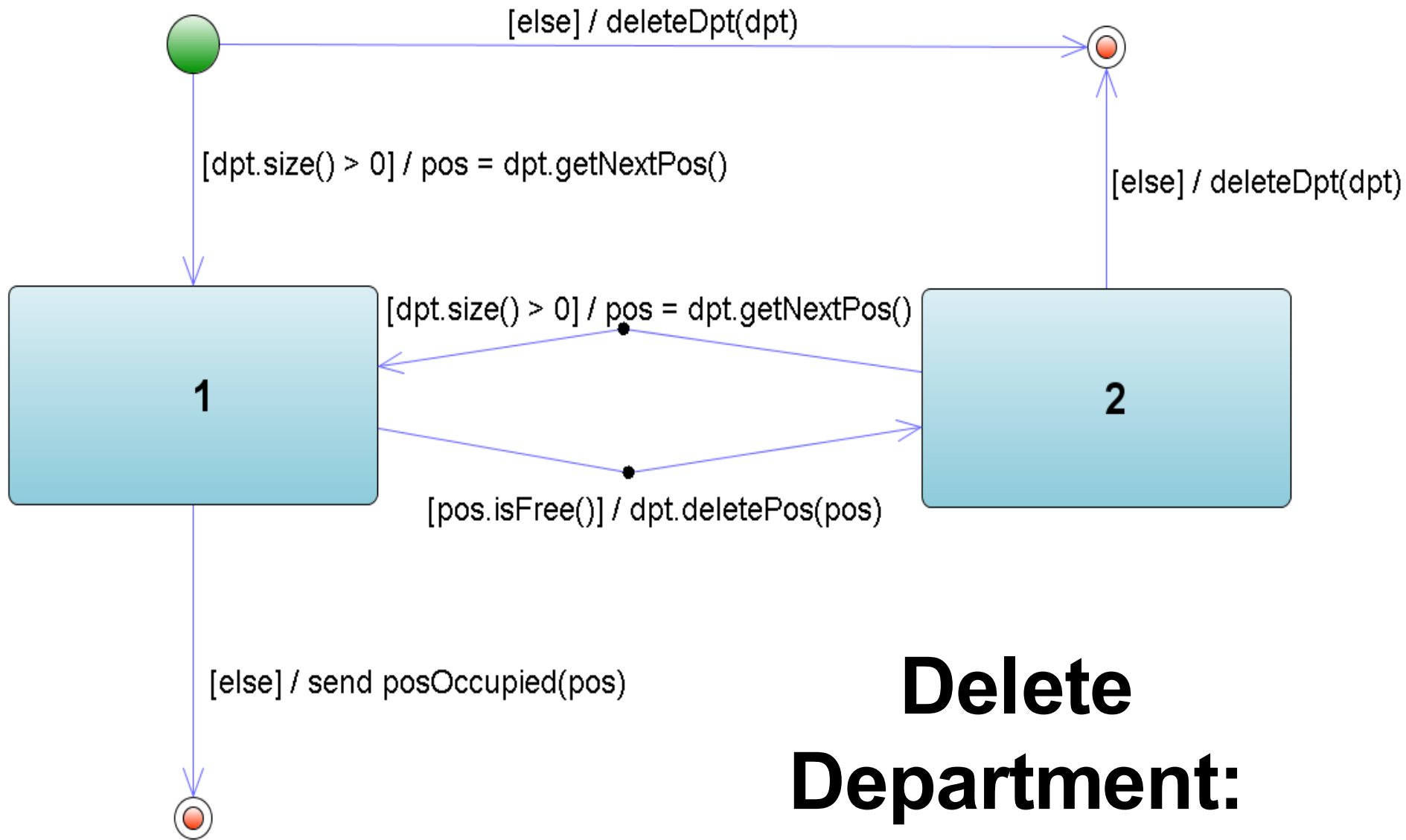
- **while dpt.size() > 0**
- **pos := dpt.getNextPos()**
- **if not pos.isFree() then**
- **send posOccupied(pos)**
- **stop**
- **dpt.deletePos(pos)**
- **deleteDpt(dpt)**

Delete Department: диаграмма (3)



Delete Department: диаграмма (4)

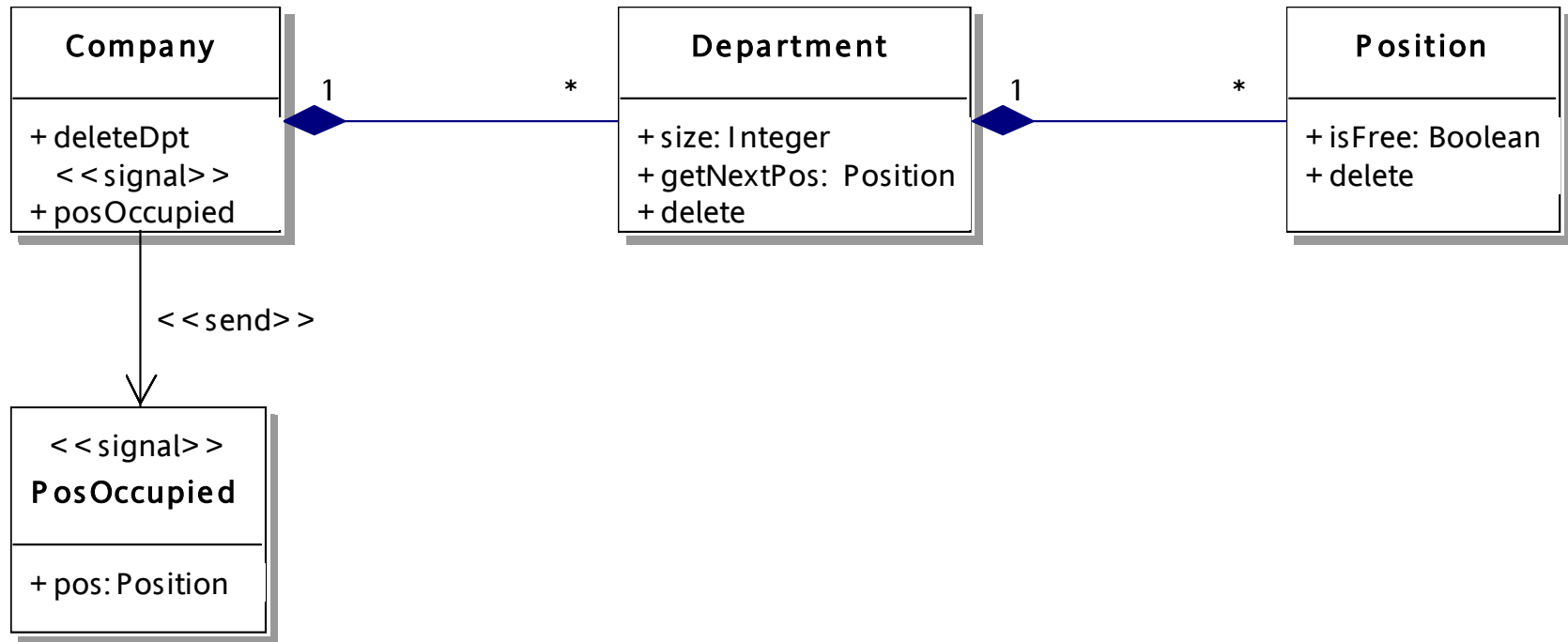


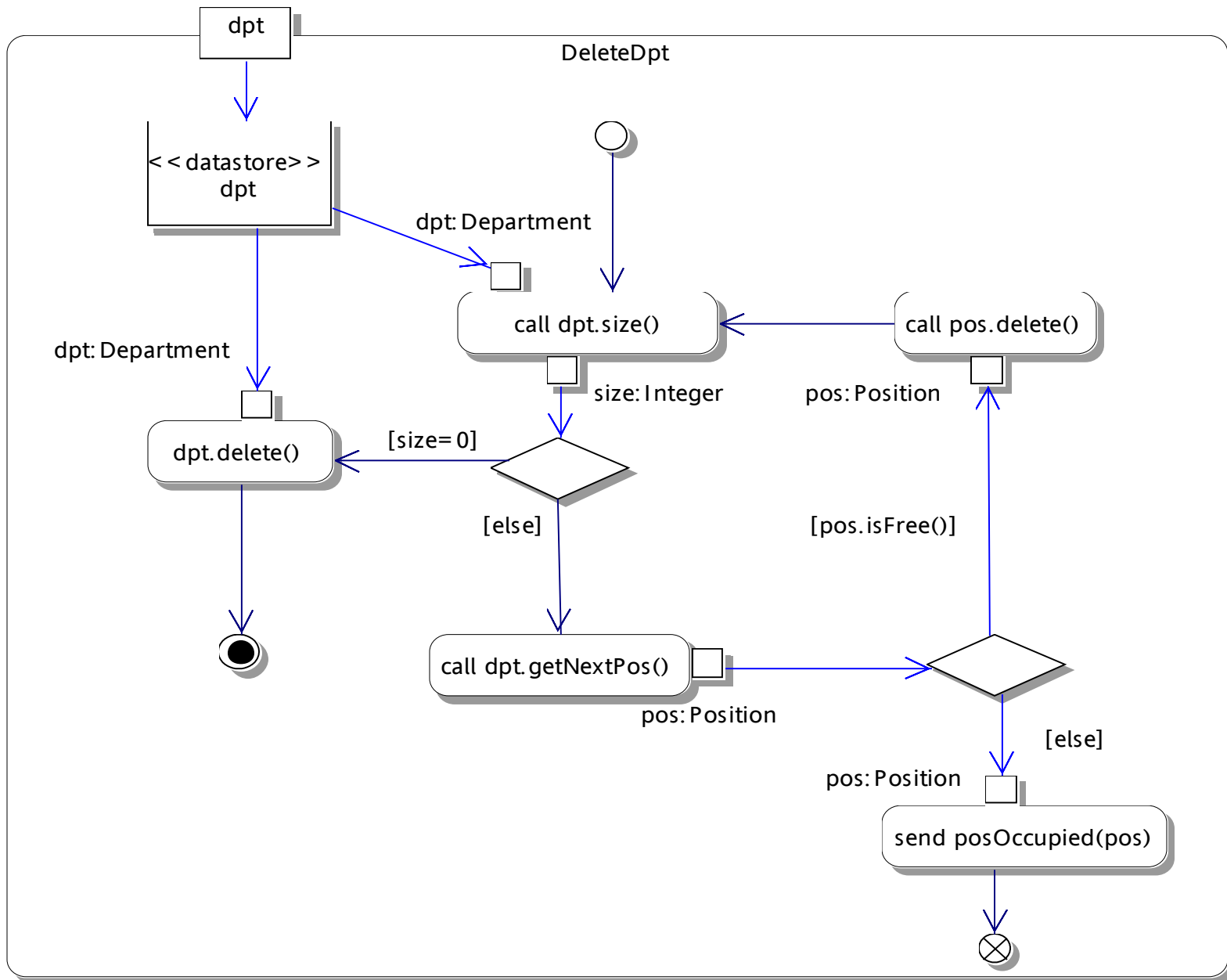


Delete Department: диаграмма (5)



Delete Department в нотации UML 2.0

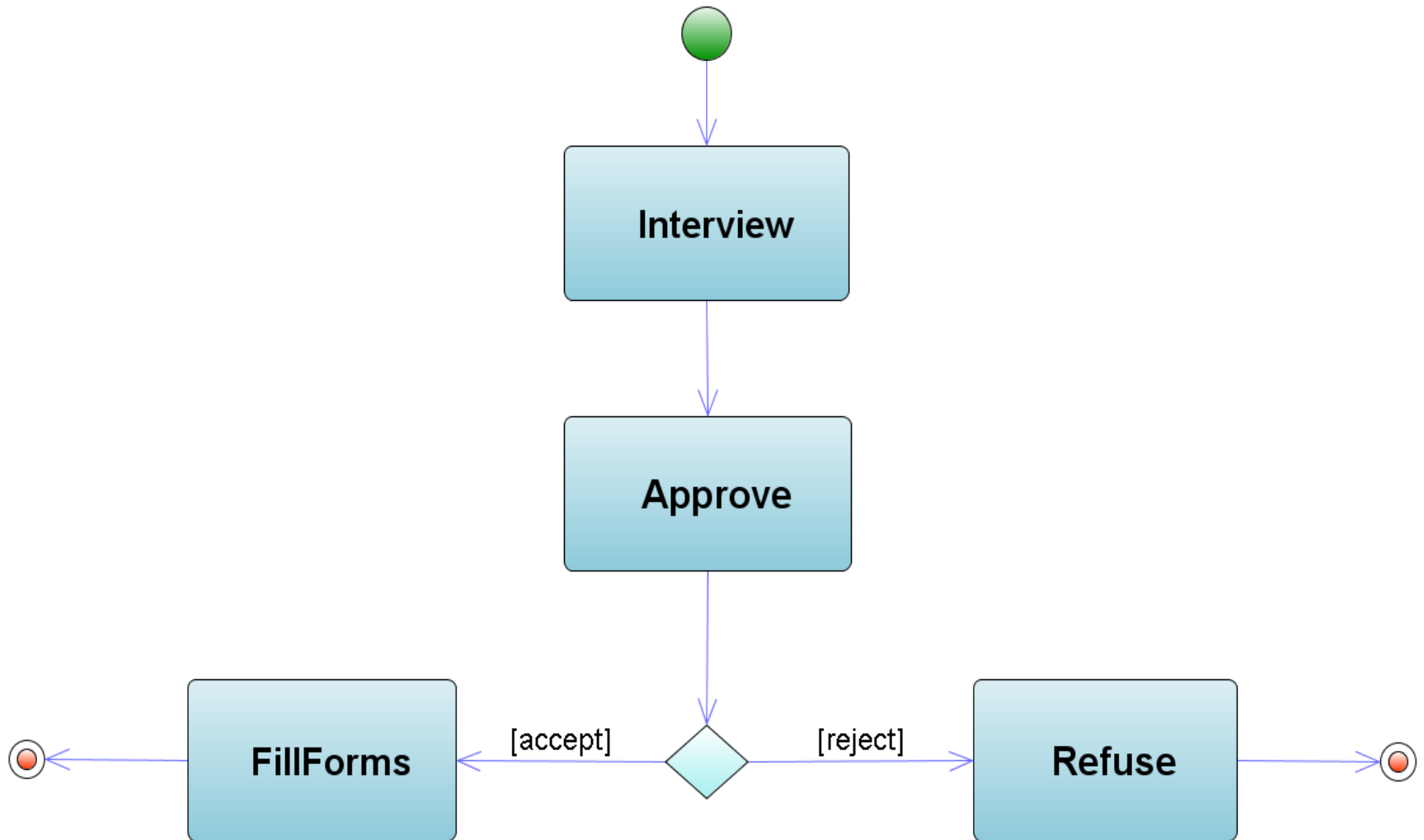




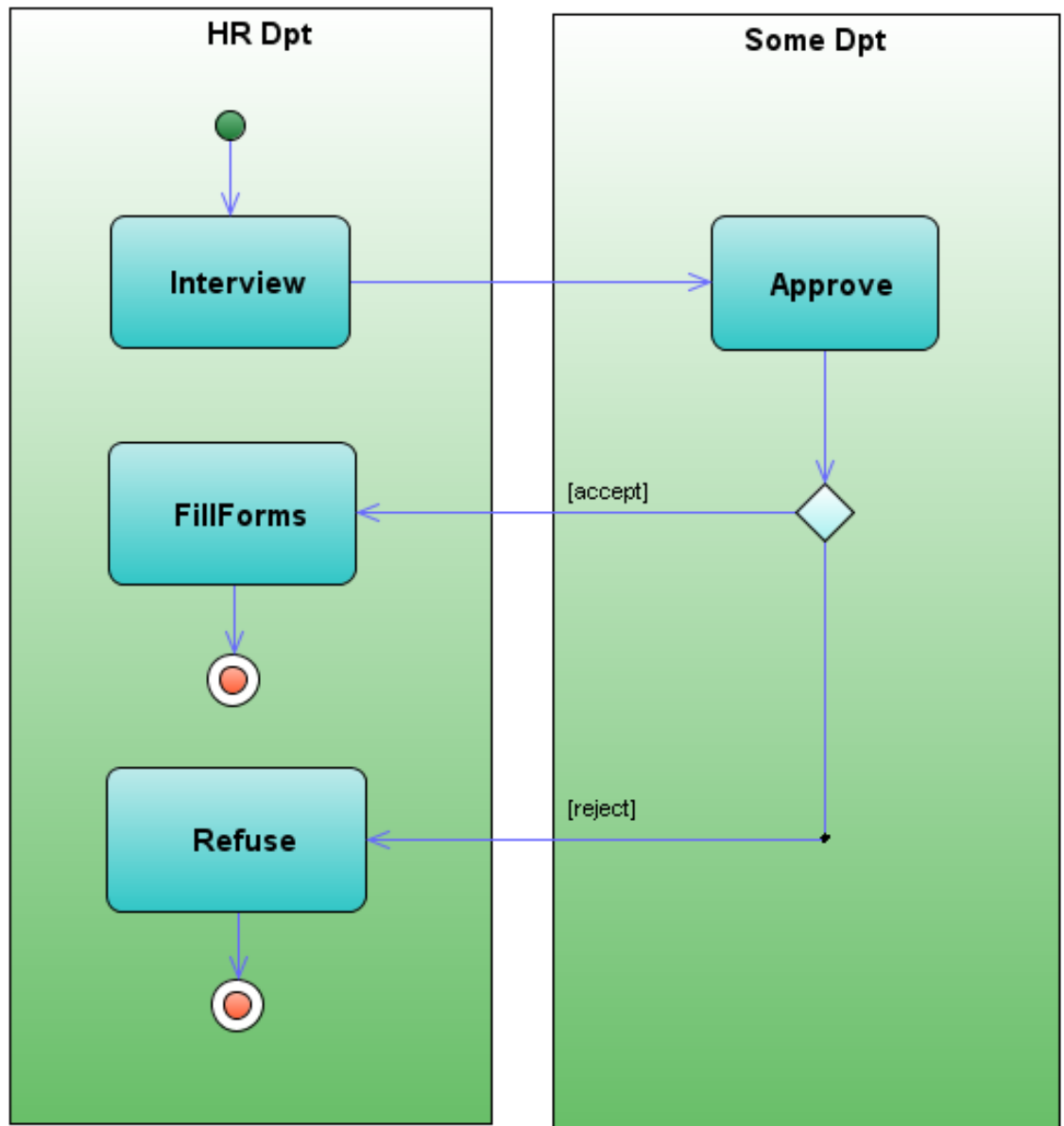
Разделы (плавательные дорожки)

- В UML 1.4 произвольная классификация действий – графический комментарий
- В UML 2.0 могут быть использованы для разнесения действий по классам
- Структурированные и ортогональные разделы в UML 2.0
 - По двум направлениям – по вертикали и по горизонтали
 - С подразделениями на более мелкие подразделы
 - Со стандартными стереотипами <<class>>, <<property>>, ...

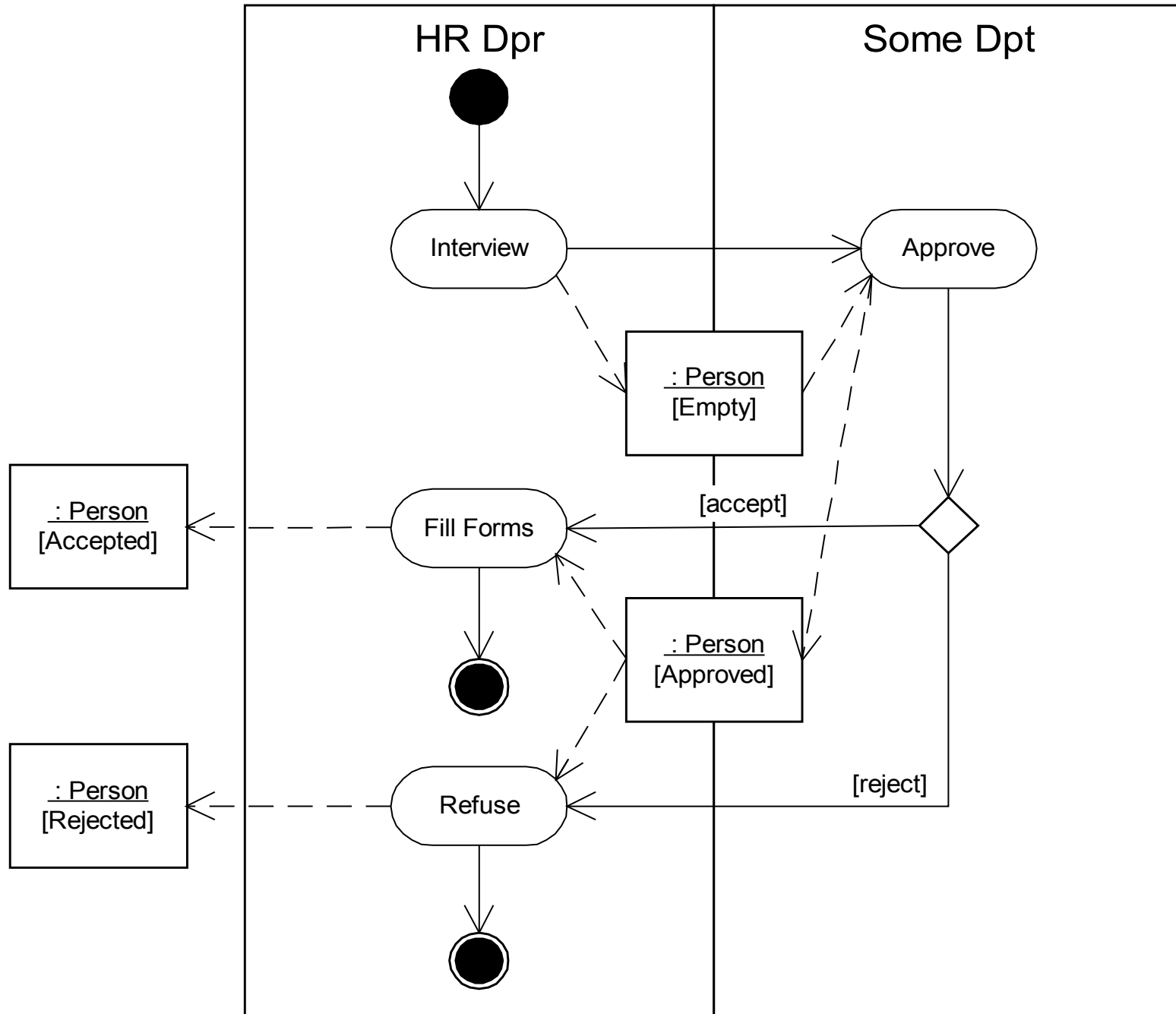
Пример ИС ОК: прием



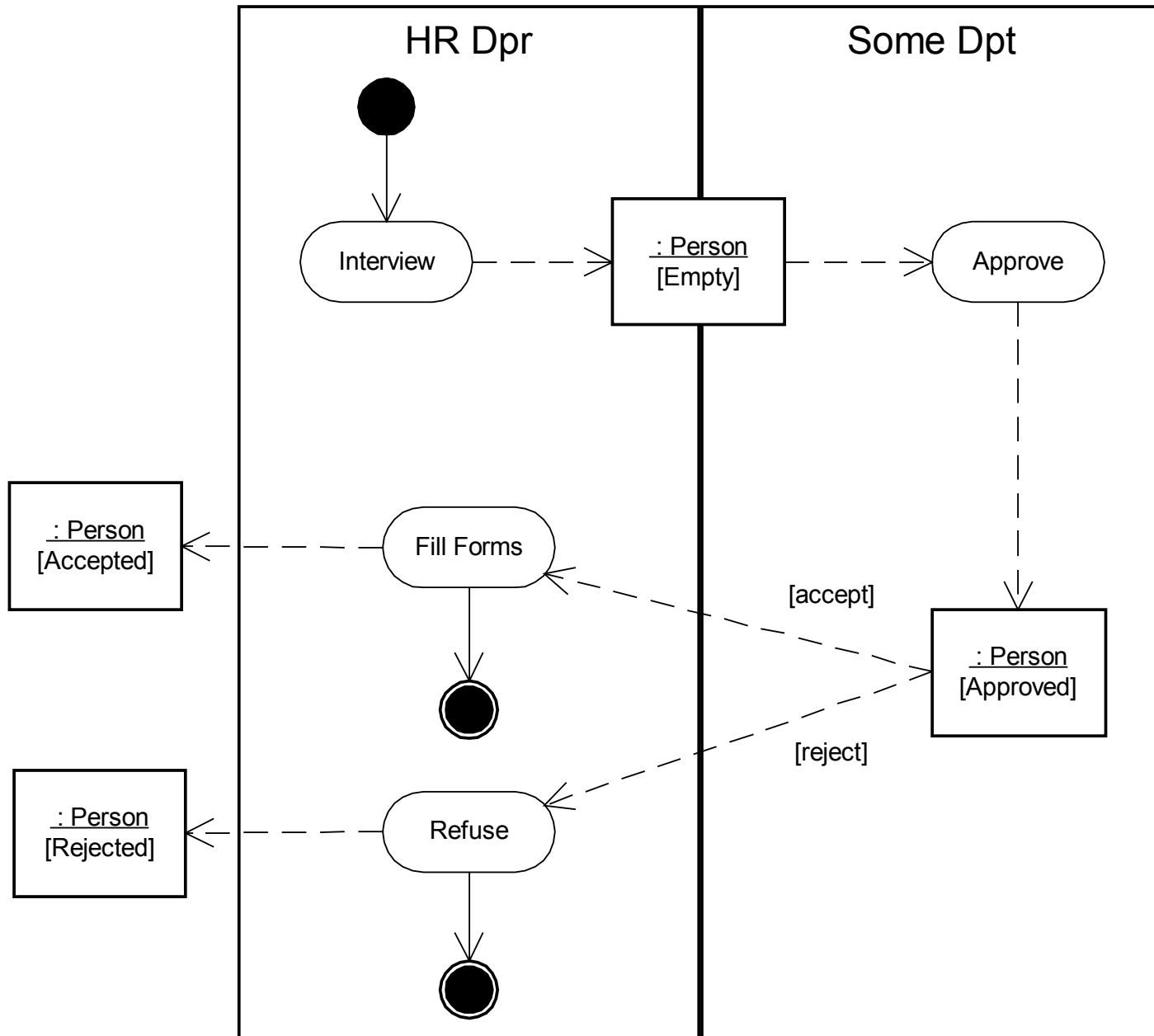
Дорожки swim lanes partitions



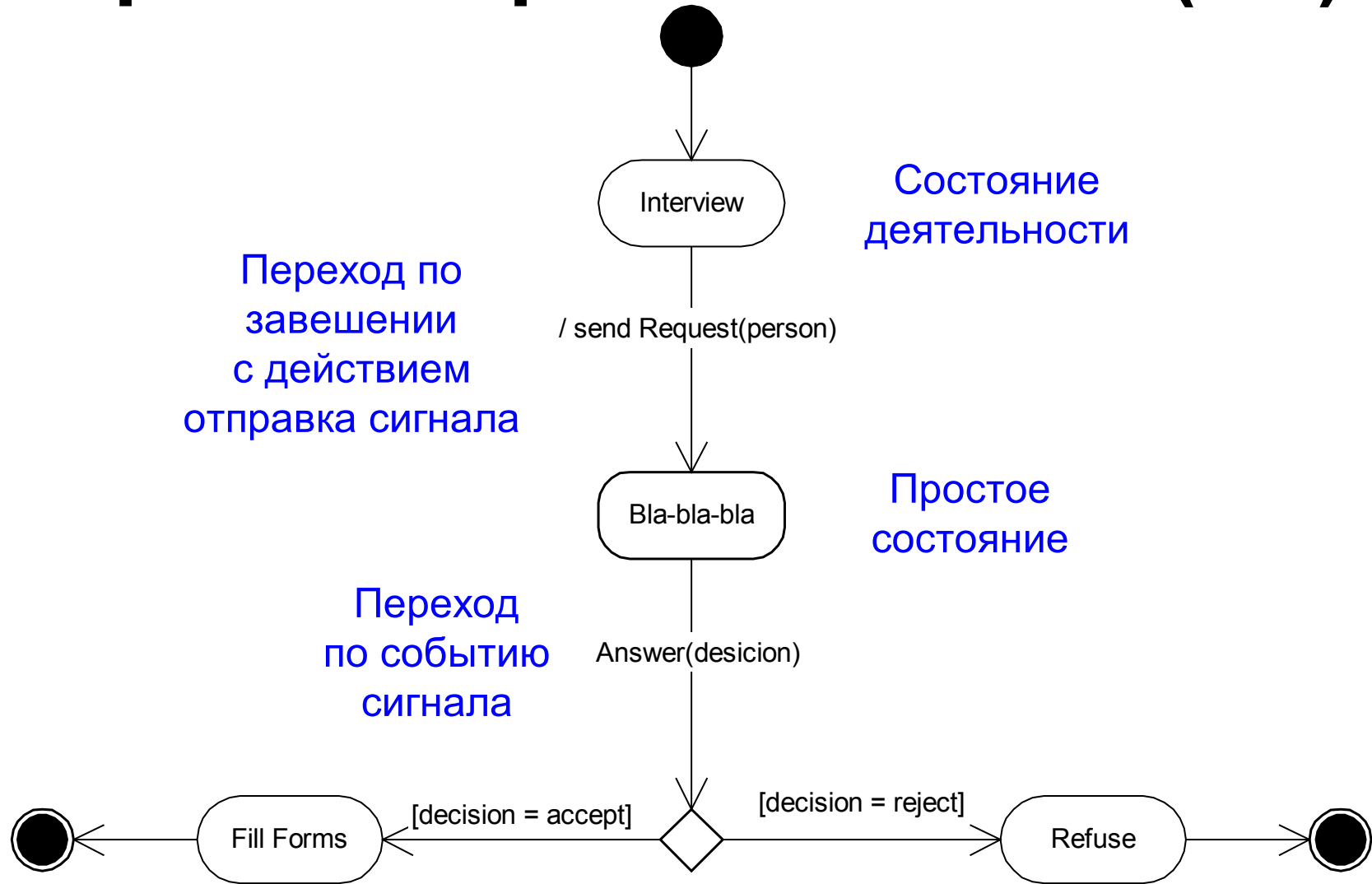
Траектория объекта (1.4)



Траектория объекта (1.4)



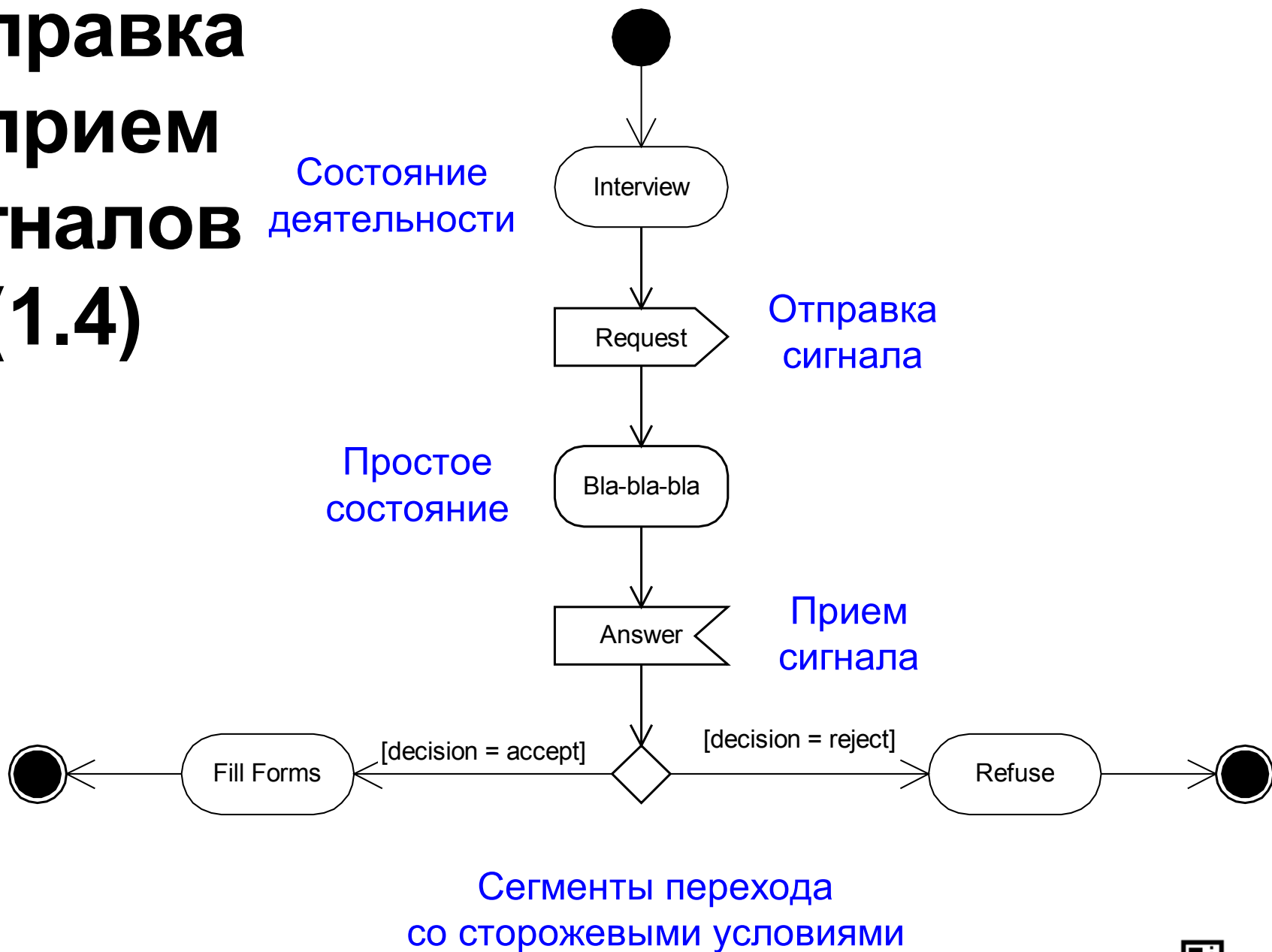
Отправка и прием сигналов (1.4)



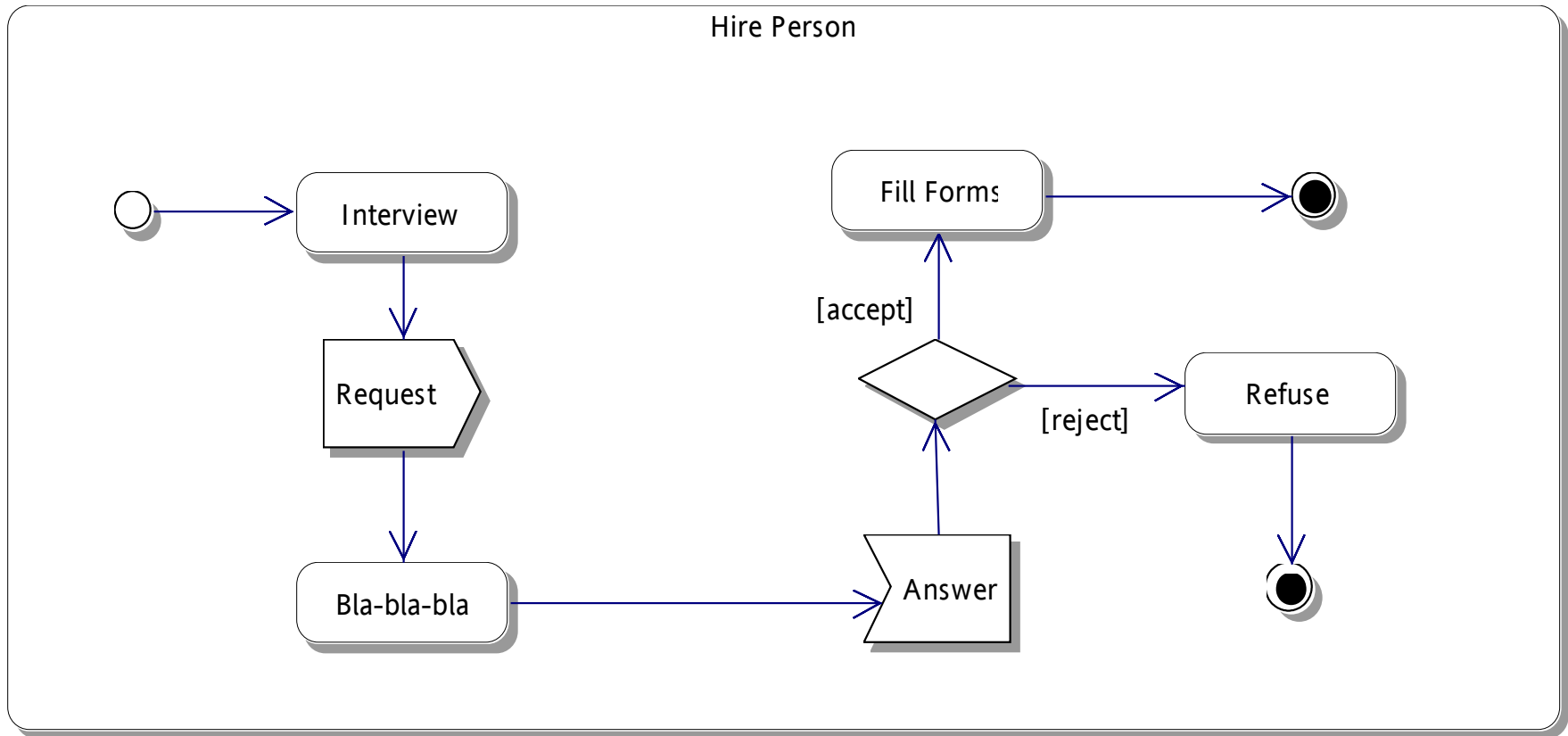
Сегменты перехода
со сторожевыми условиями



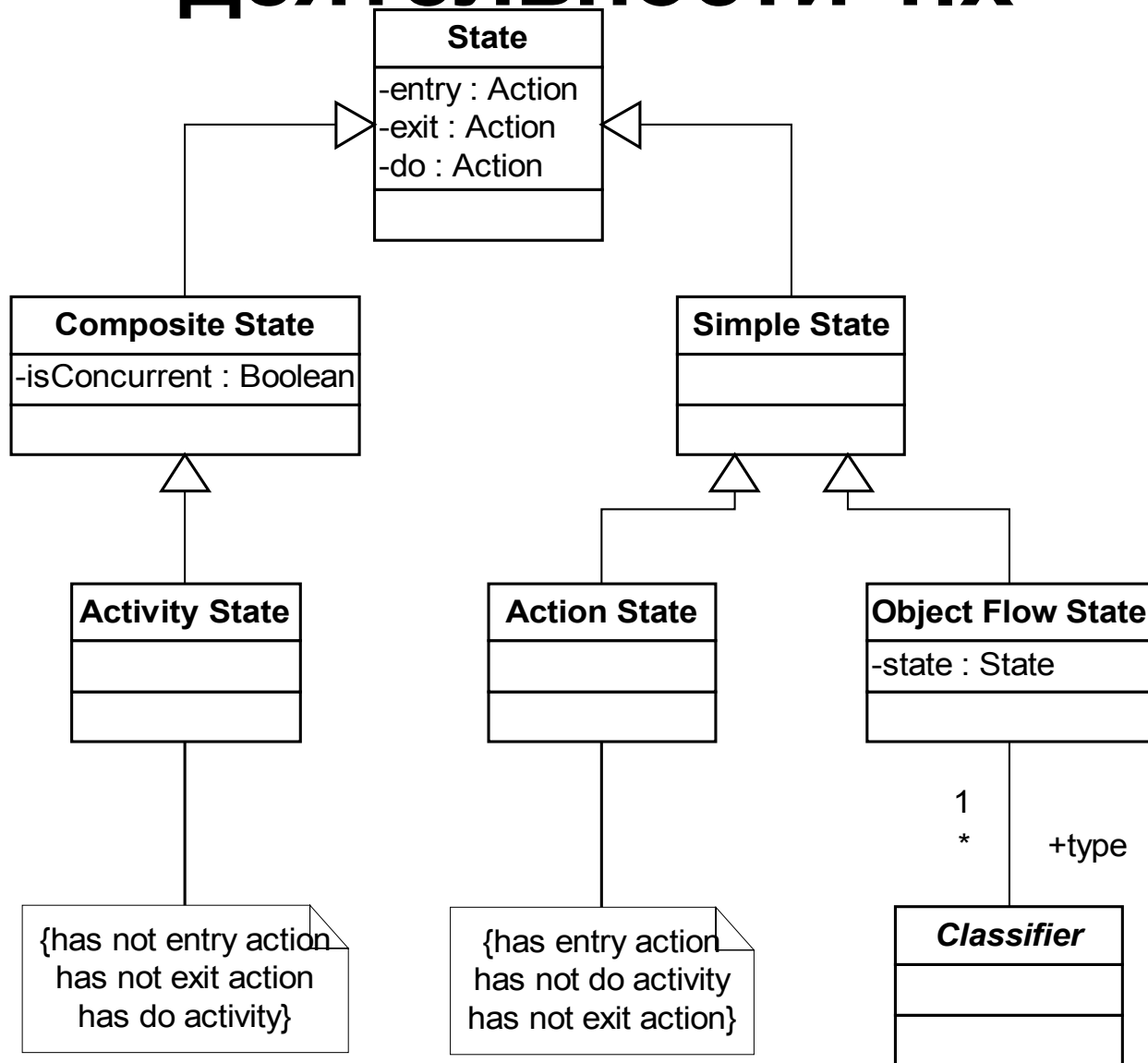
Отправка и прием сигналов (1.4)



Отправка и прием сигналов в UML 2.0



Метамодель элементов диаграммы деятельности 1.x



5.4. Диаграммы взаимодействия

- **Диаграммы коммуникации и диаграммы последовательности семантически эквиваленты**
- **Объекты: экземпляры классификаторов – классов и действующих лиц (а также узлов и компонентов)**
- **Связи: экземпляры ассоциаций**
- **Сообщения: передаются вдоль связей**

Сообщения

- **Call – вызов операции**
 - Объект может вызвать свою операцию
 - Операция должна быть определена
- **Return – возврат значения**
 - Показывать отдельно не обязательно
- **Send – посылка сигнала**
 - Взаимодействие автоматов
- **Create – создание объекта**
 - Вызов конструктора
- **Destroy – уничтожение объекта**
 - Объект может уничтожить себя

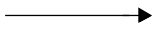

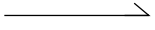

Нотация сообщений на диаграммах коммуникации (1)

- Стрелка вверх связи
- Тело сообщения
 - предшественники / повторность номер :
 - «стереотип» переменные := ИМЯ (аргументы)
- Предшественники
 - Список номеров сообщений, предшествующих данному
- Повторность
 - [Сторожевое условие]
 - * [Итерация сообщений] = broadcasting

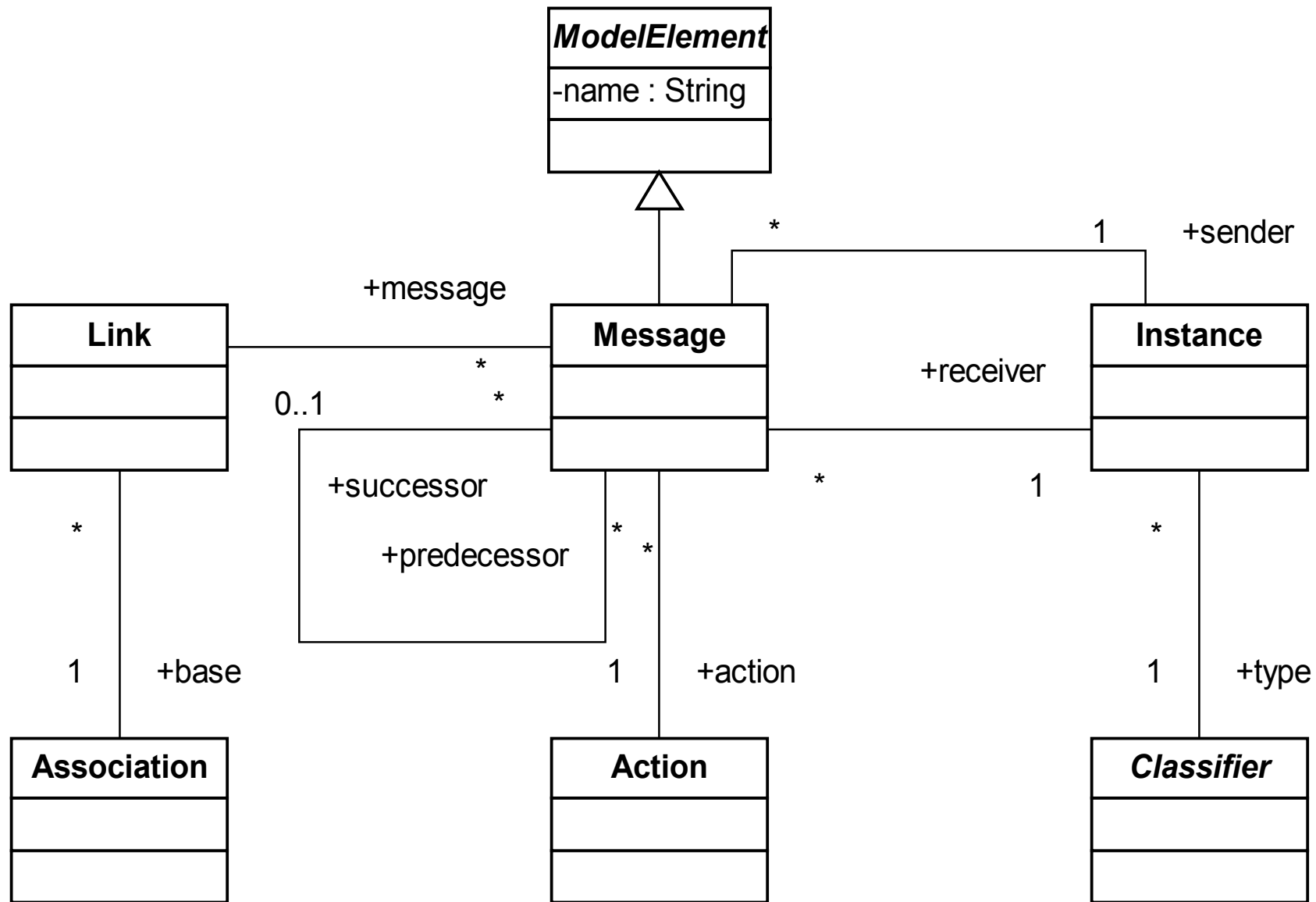
Нотация сообщений на диаграммах коммуникации (2)

- **Стереотип**
 - «call» «create» и т.д. – не обязателен
- **Номер сообщения**
 - Иерархическая десятичная нумерация – отражает вложенность потоков управления
 - Если параллельные потоки, то буквы:
А, В, ...
- **Переменные**
 - Для возвращаемых значений

Типы передачи сообщений

Стрелка	Тип передачи
	Вложенный поток управления
	Простой поток управления
	Асинхронный поток управления
	Возврат управления
Не опр.	Может быть добавлен инструментом

Метамодель сообщения



5.5. Диаграммы последовательности

- Линия жизни объекта (life line)
- Метка времени (timestamp)
- Ограничения на время
- Задержанная доставка сообщения
- В UML 2.0 появились разновидности диаграмм последовательности:
 - Составные шаги (Combined Fragments)
 - Диаграмма синхронизации (Timing Diagram)
 - Обзорная диаграмма взаимодействия (Interaction Overview Diagram)

Пример ИС ОК: создание подразделения

Экземпляр
действующего
лица

Постоянно
существующий
объект

Первое
сообщение

Сообщение

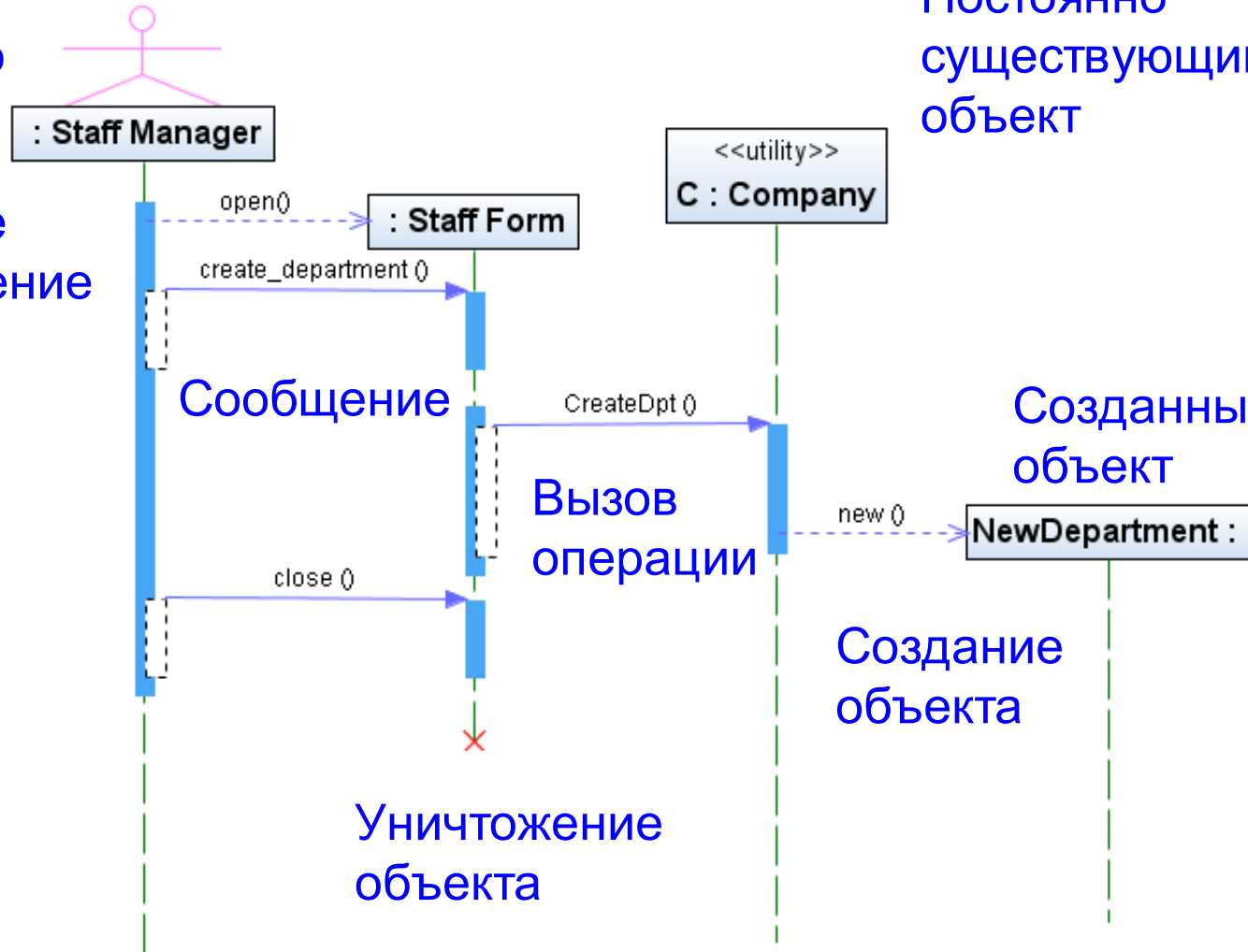
Вызов
операции

Созданный
объект

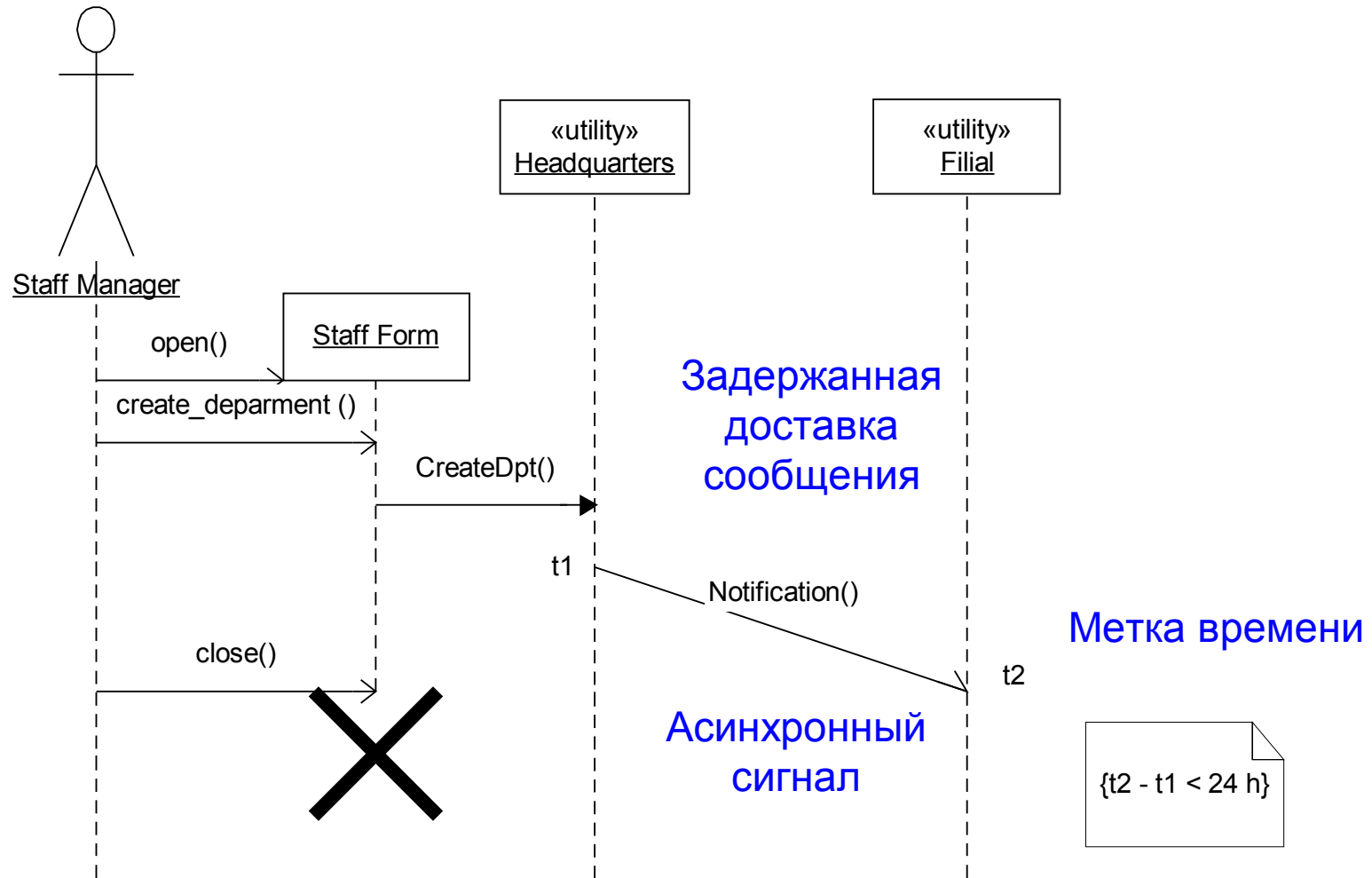
Создание
объекта

Уничтожение
объекта

Линия жизни

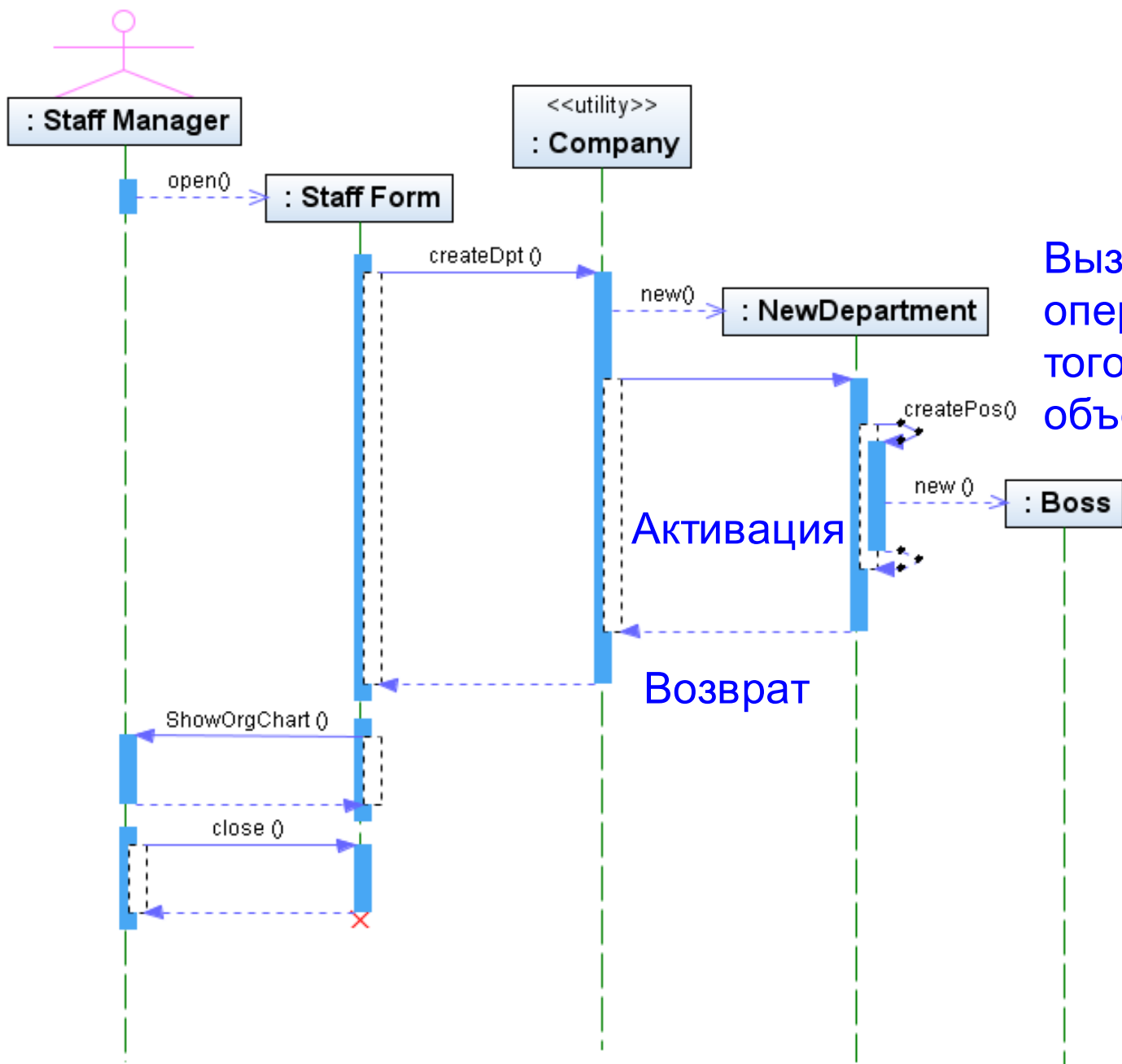


Задержанная доставка сообщений



Ограничение

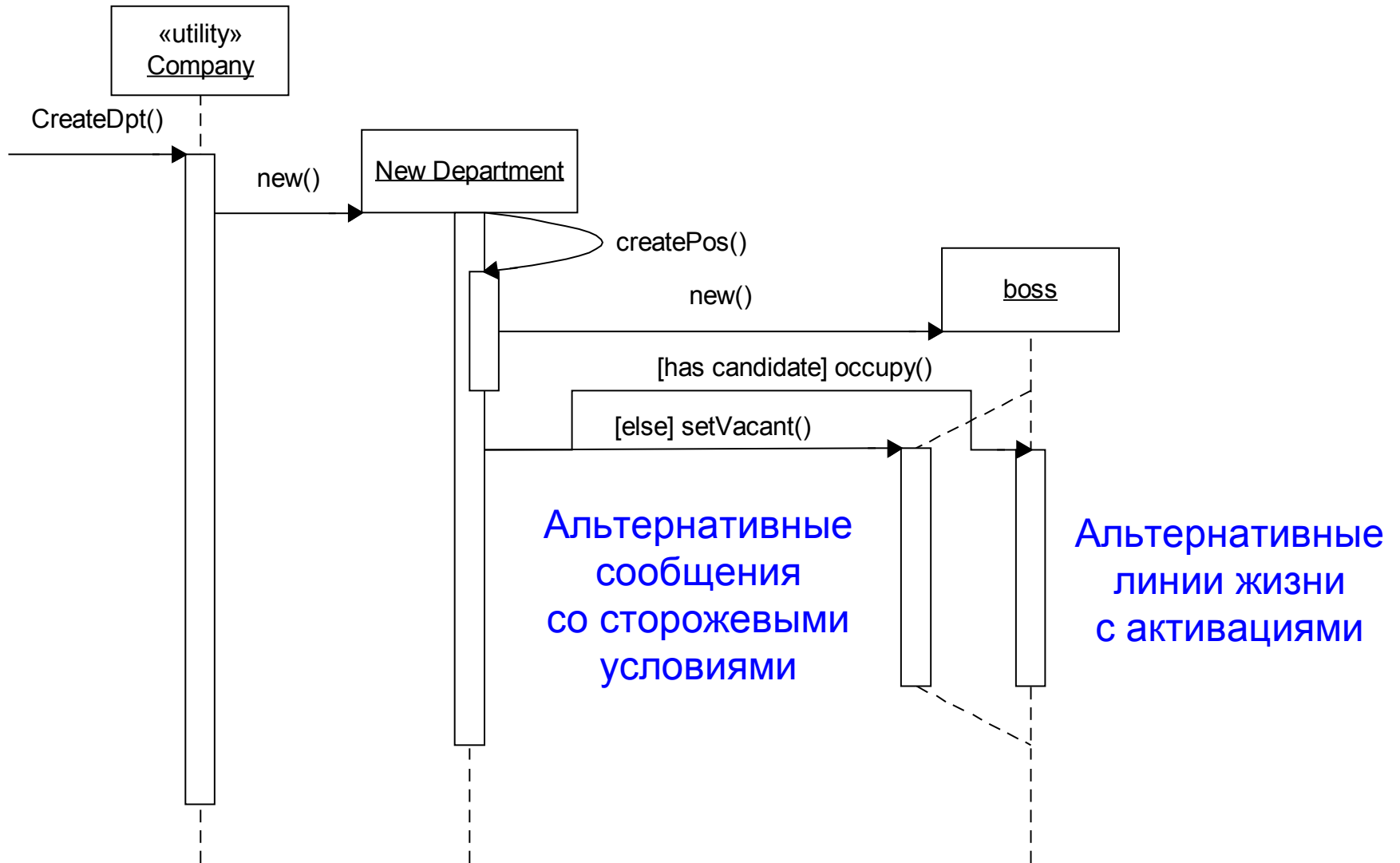
Активация и возврат



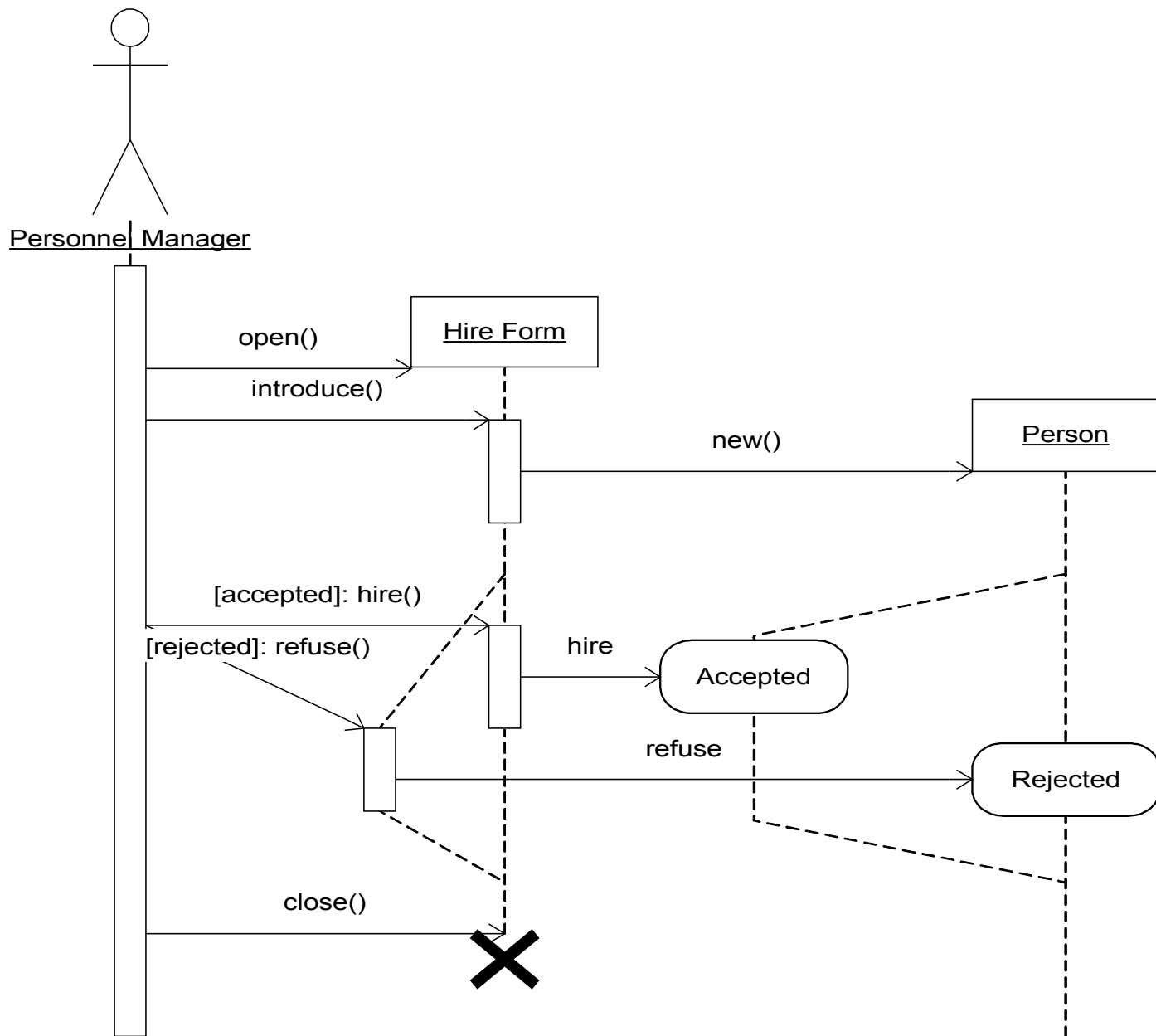
Вызов
операции
того же
объекта



Ветвления (1.x)



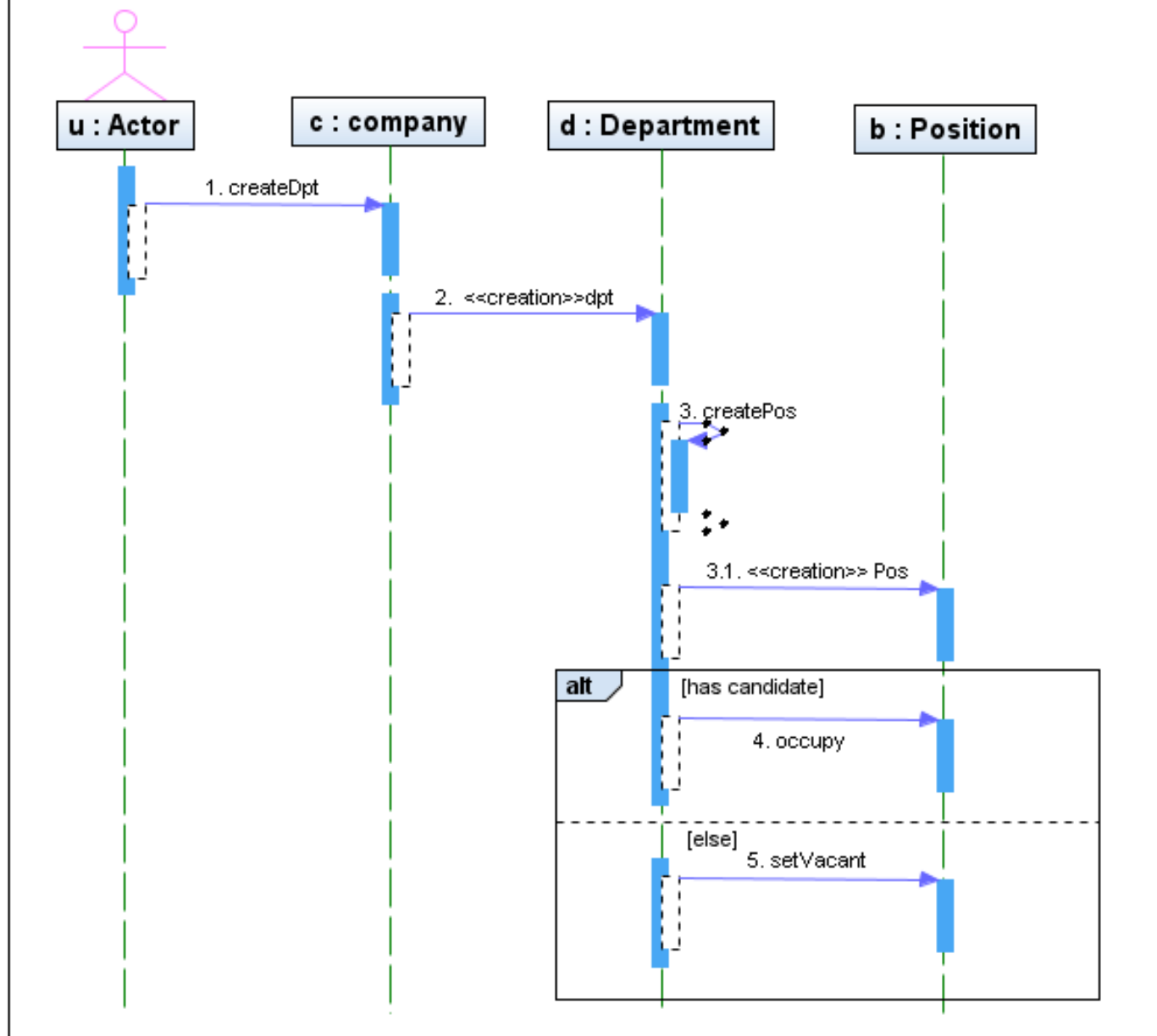
Состояния объекта на диаграмме последовательности (1.x)



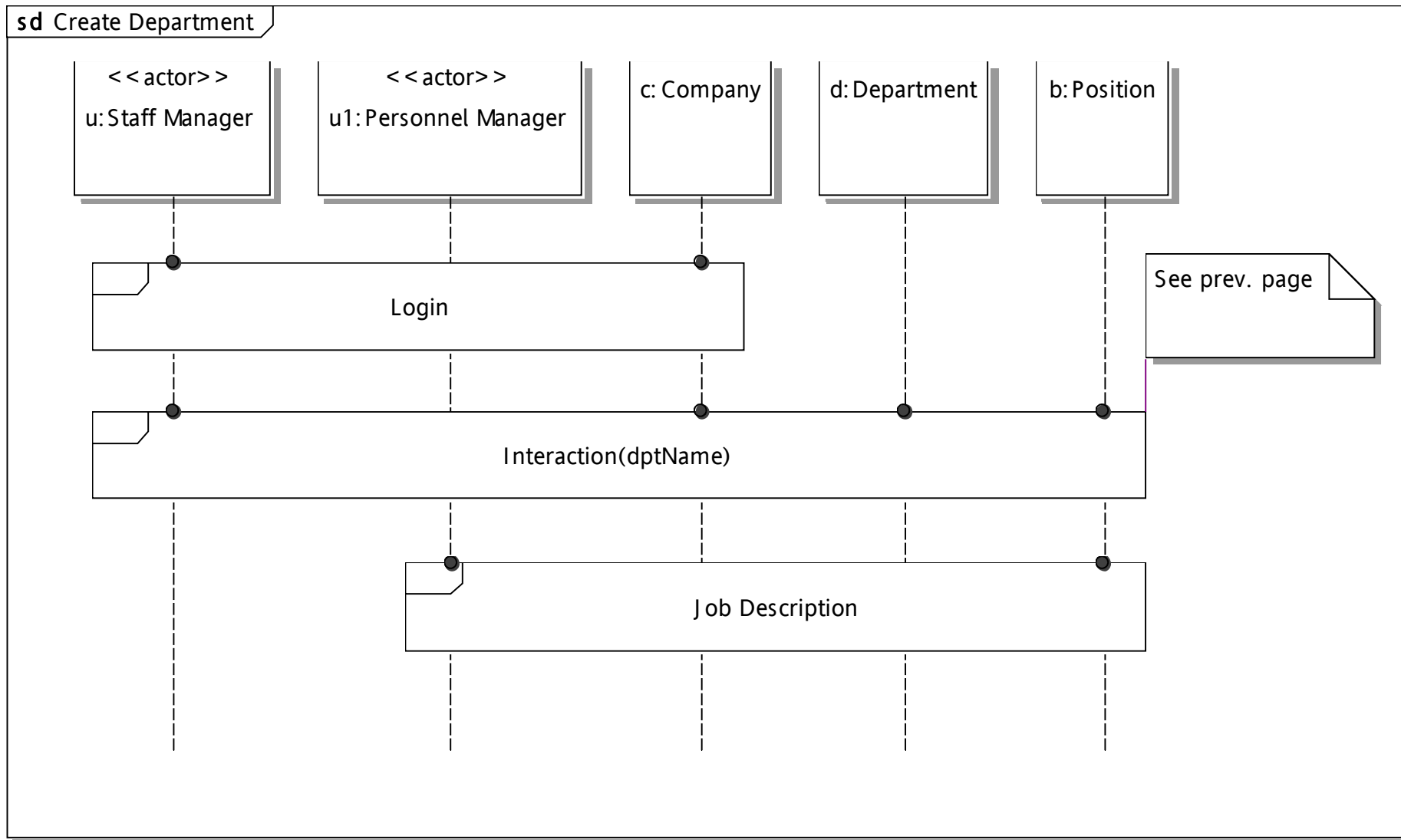
Составные шаги взаимодействия

Тег	Формат	Описание
alt	Условие в каждом шаге	Выбор одной альтернативы
assert	Ограничение в шаге	Проверка инварианта
break	Условие в шаге	Прекращение супер шага
consider	Список сообщений	Рассматриваемые сообщения
critical		Другие сообщения запрещаются
ignore	Список сообщений	Игнорируемые сообщения
loop	Кратность и условие	Циклическое повторение шага
neg		Запрещенная последовательность
opt	Условие	Условное выполнение шага
par		Параллельное выполнение шагов
seq		Слабое упорядочение шагов
strict		Сильное упорядочение шагов

sd Interaction



Ссылка на взаимодействие (interaction use)



Обзорная диаграмма взаимодействия (Interaction Overview Diagram)

- Простая и полезная идея из MSC
- Диаграмма взаимодействия в форме диаграммы деятельности
- Вместо действий – диаграммы взаимодействия (последовательности или коммуникации) (или ref)
- Ветвления и развилки
- Не поддерживается в Sun Java Studio Enterprise 8

Диаграмма синхронизации (Timing Diagram)

- Простая и полезная идея из схемотехники
- По оси абсцисс – время (в масштабе)
- По оси ординат – полоса жизни, подразделенная на состояния
- Линия жизни – ломаная
 - Горизонтальный отрезок – объект в состоянии
 - Вертикальный (или наклонный!) отрезок – смена состояния
 - Около вертикального отрезка сообщение, вызывающее изменение состояния
- **Не поддерживается в
Sun Java Studio Enterprise 8**

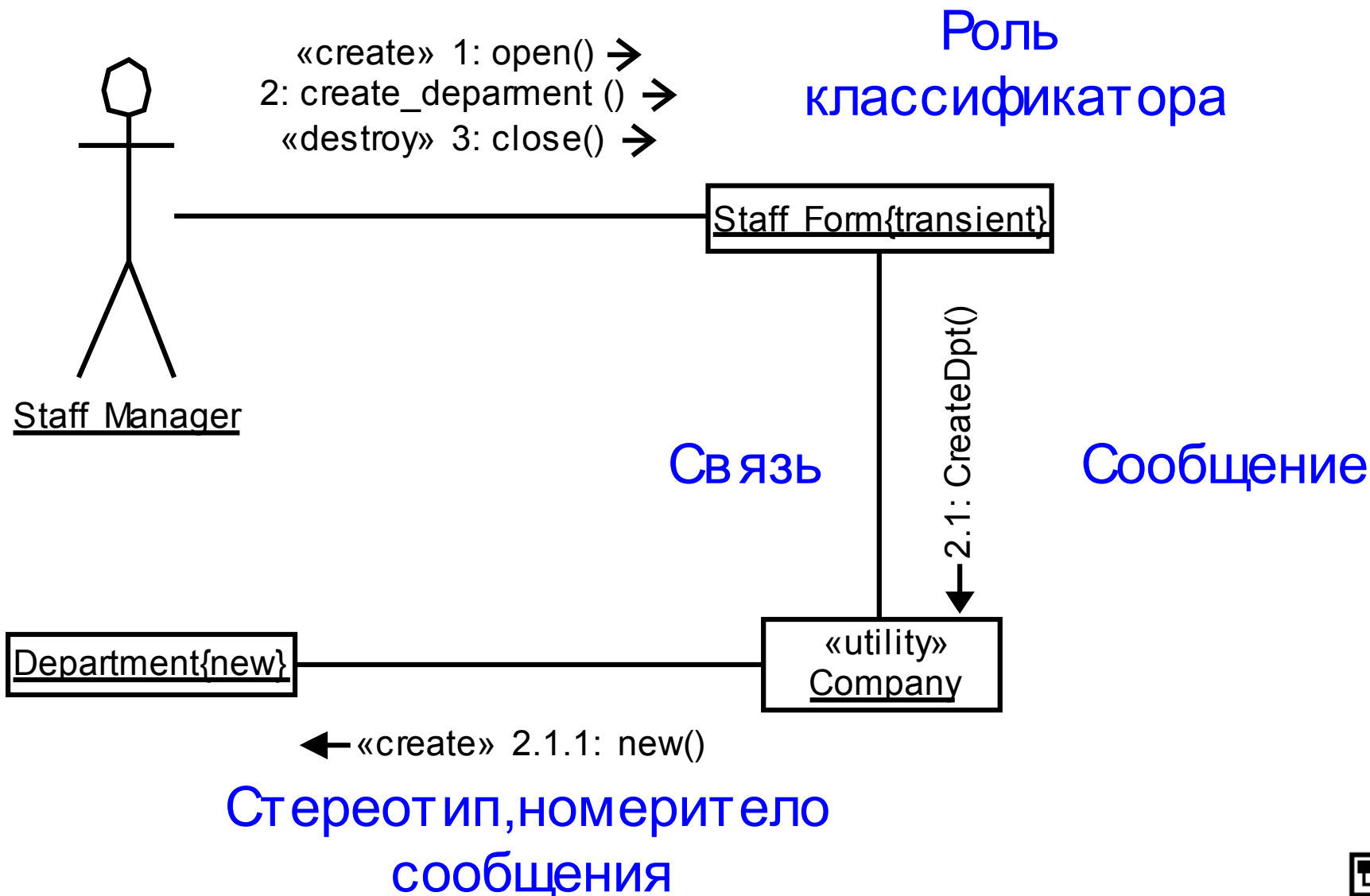
5.6. Диаграммы коммуникации и кооперации

- Уточнение понятий при сохранении общей концепции
- **1.x роль классификатора** = слот объекта → **2.0 роль** (часть)
- **1.x роль ассоциации** = слот связи → **2.0 роль** (соединитель)
- **1.x контекст взаимодействия** = диаграмма объектов без сообщений → **2.0 кооперация**
- Кооперация м.б. описана
 - В структурированном классификаторе
 - В качестве образца проектирования
 - В любом контексте (например, вариант использования)
- Кооперация является классификатором (но не классом!)
- Кооперация, нагруженная сообщениями, образует диаграмму коммуникации

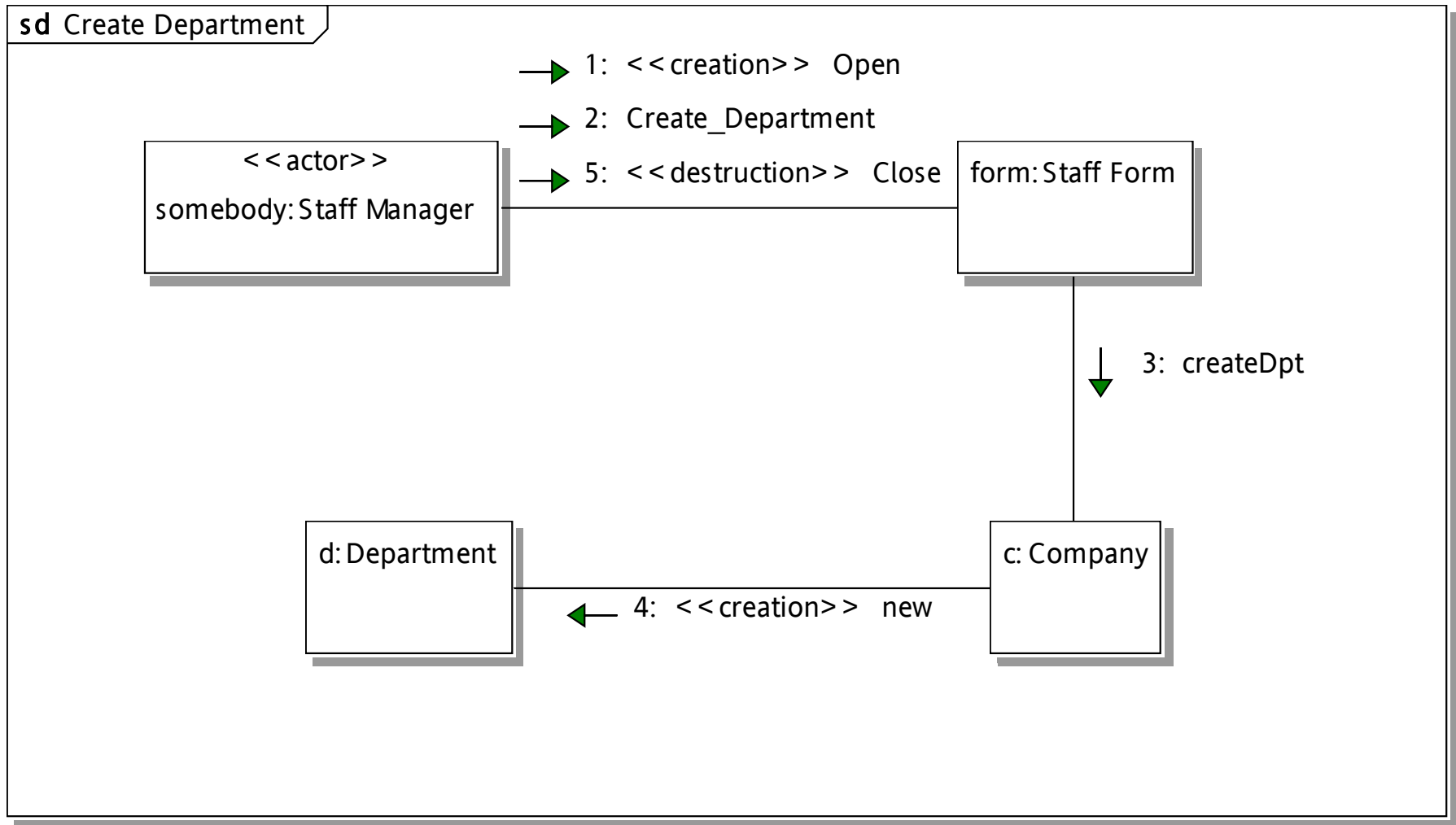
Время жизни объектов (1.x)

Ключевое слово	Способ использования	Описание
<code>create</code>	стереотип операции в сообщении	Операция создает объект, т. е. данное сообщение является вызовом конструктора
<code>destroy</code>	стереотип операции в сообщении	Операция уничтожает объект, т. е. данное сообщение является вызовом деструктора
<code>destroyed</code>	ограничение роли классификатора	Объект уничтожается в процессе описываемого взаимодействия
<code>new</code>	ограничение роли классификатора	Объект создается в процессе описываемого взаимодействия
<code>transient</code>	ограничение роли классификатора	Объект создается и уничтожается в процессе описываемого взаимодействия.

Create Department (1.x)



Create Department (2.0)



Стереотипы полюса ассоциации (1.x)

Стереотип

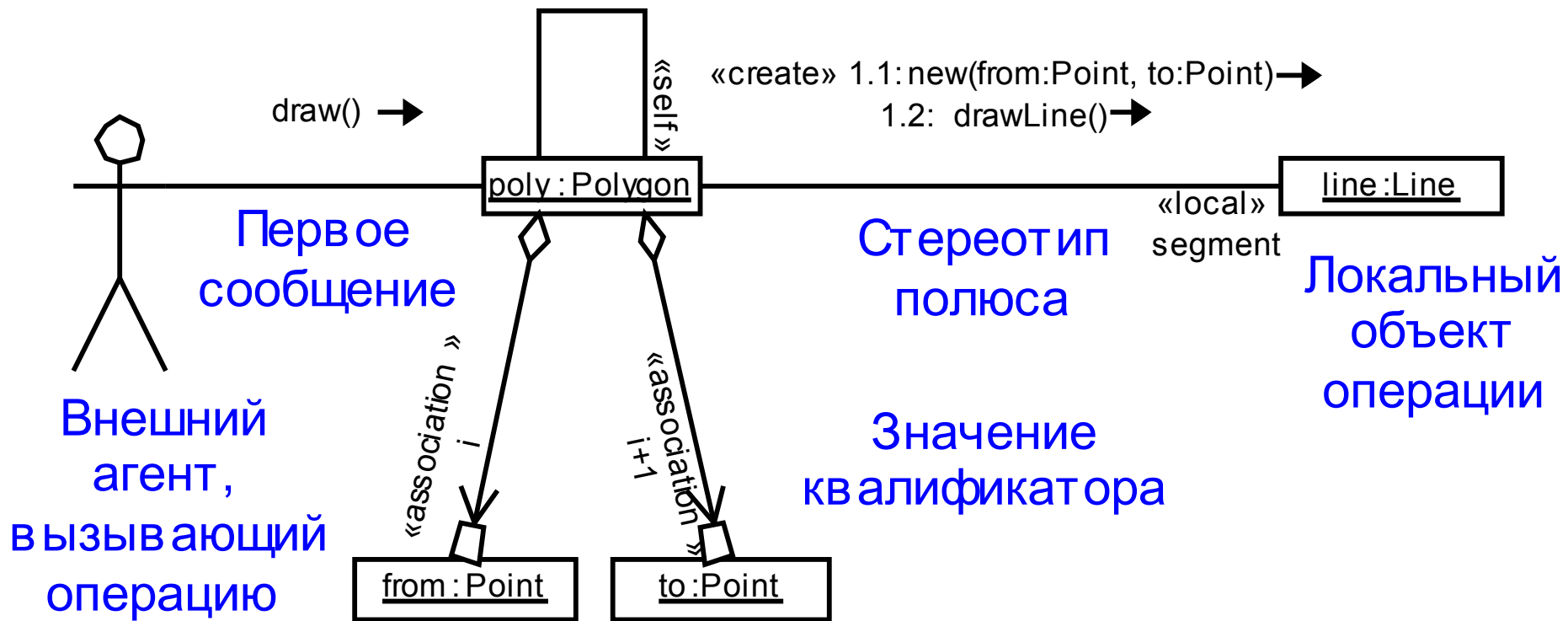
Описание

«association»	Объект на полюсе роли ассоциации связан с объектом на противоположном полюсе фактической связью, реализующей ассоциацию
«global»	Объект на полюсе роли ассоциации имеет глобальную область определения относительно объекта на противоположном полюсе
«local»	Объект на полюсе роли ассоциации имеет локальную область определения относительно объекта на противоположном полюсе, т. е. является временным объектом для выполнения операции
«parameter»	Объект на полюсе роли ассоциации является параметром операции объекта на противоположном полюсе
«self»	Роль ассоциации является фиктивной связью, введенной для моделирования вызова операции данного объекта

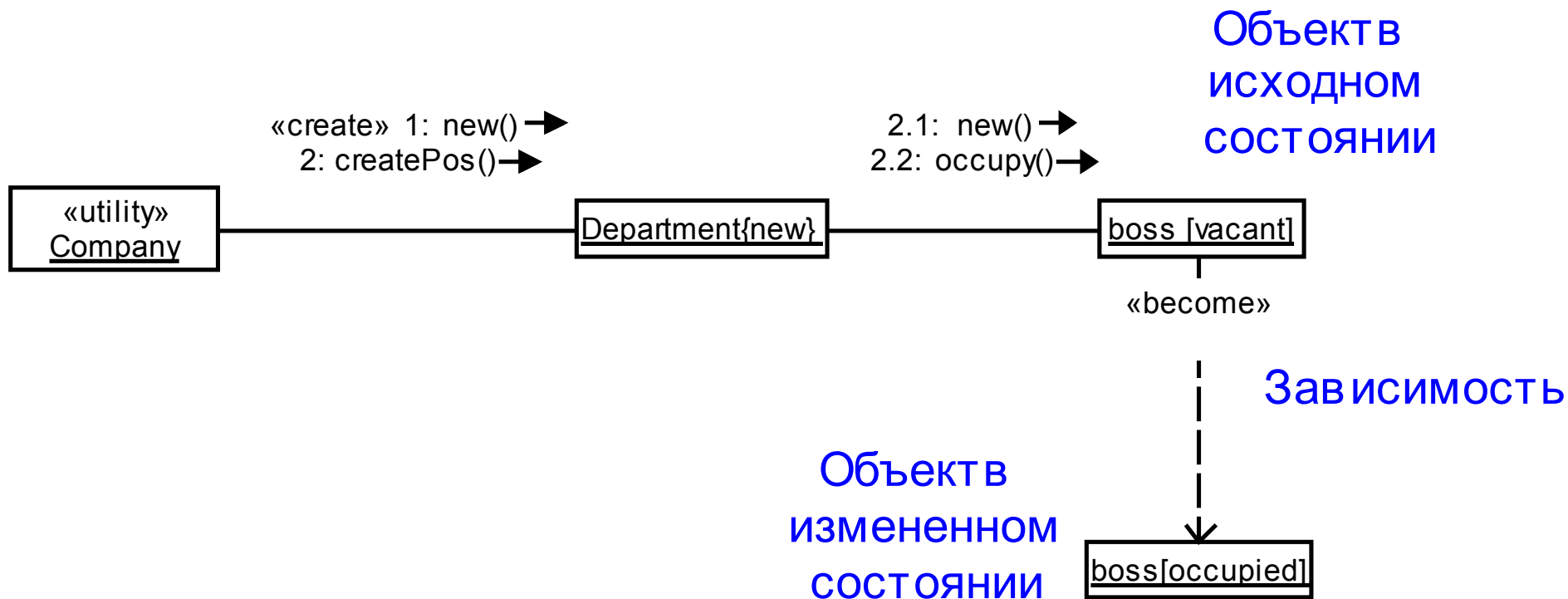
Пример: рисование многоугольника

Сообщение с повторителем

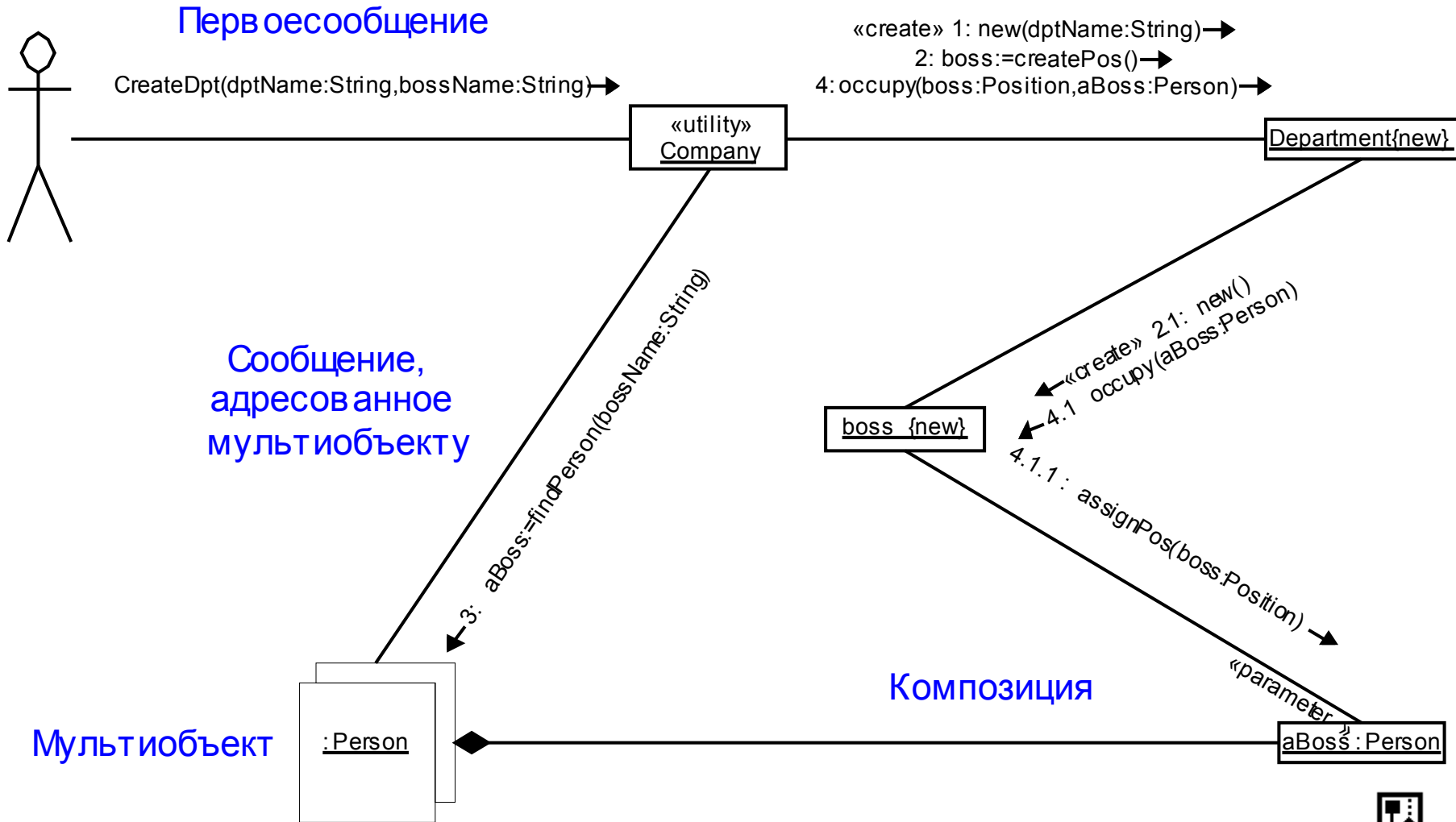
1 *[i:=1..n-1]: drawSegment(i:Integer) →



Зависимости на диаграмме кооперации (1.x)



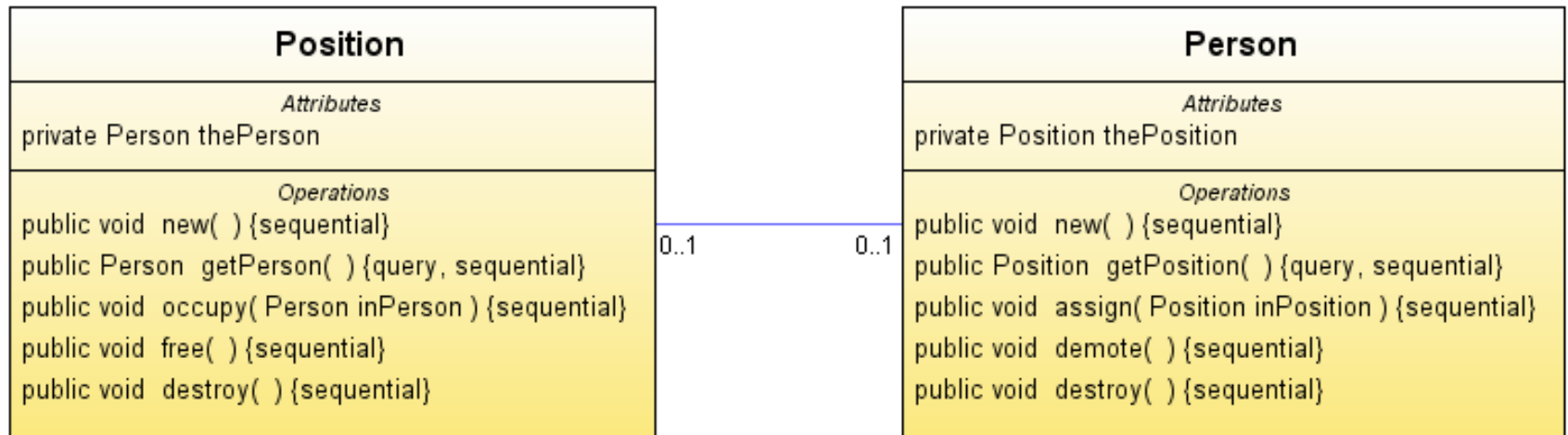
Мультиобъекты (1.x, 2.0 📄)



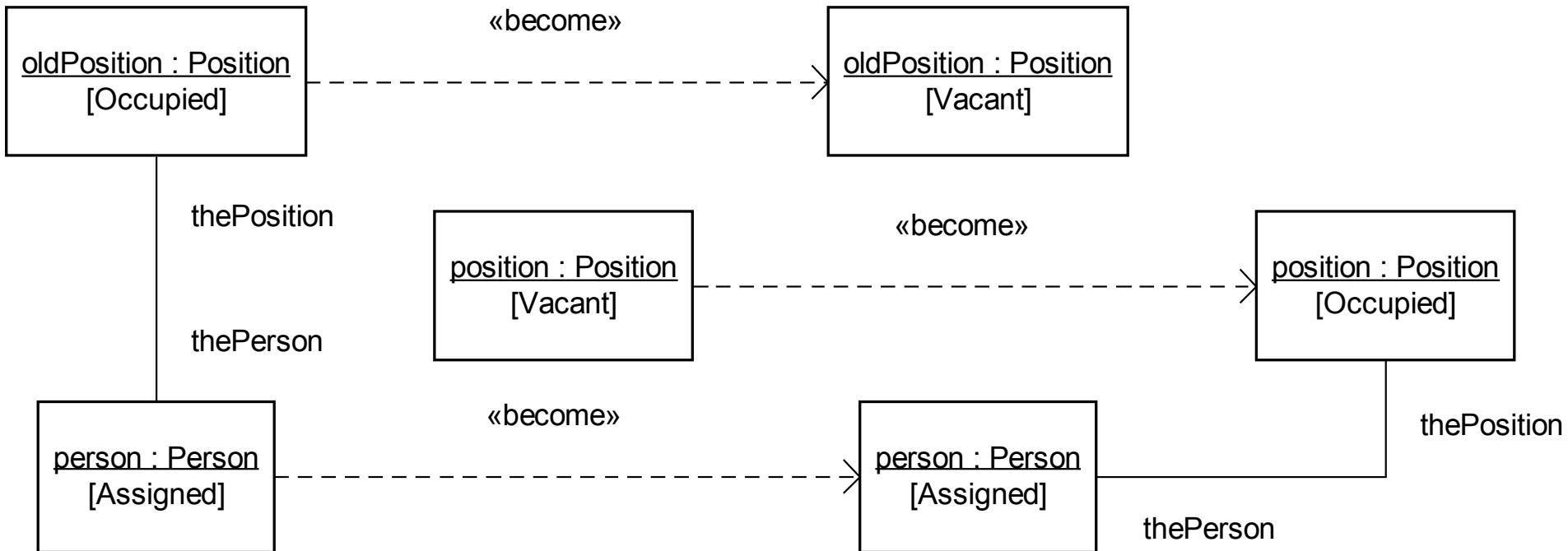
5.7. Моделирование параллелизма

- Параллелизм (concurrency) = взаимодействие последовательных процессов
- Параллельные состояния и составные переходы
- Развилки, соединения и обусловленные потоки управления
- События и асинхронные сигналы
- Потоки и процессы
- Активные классы и объекты

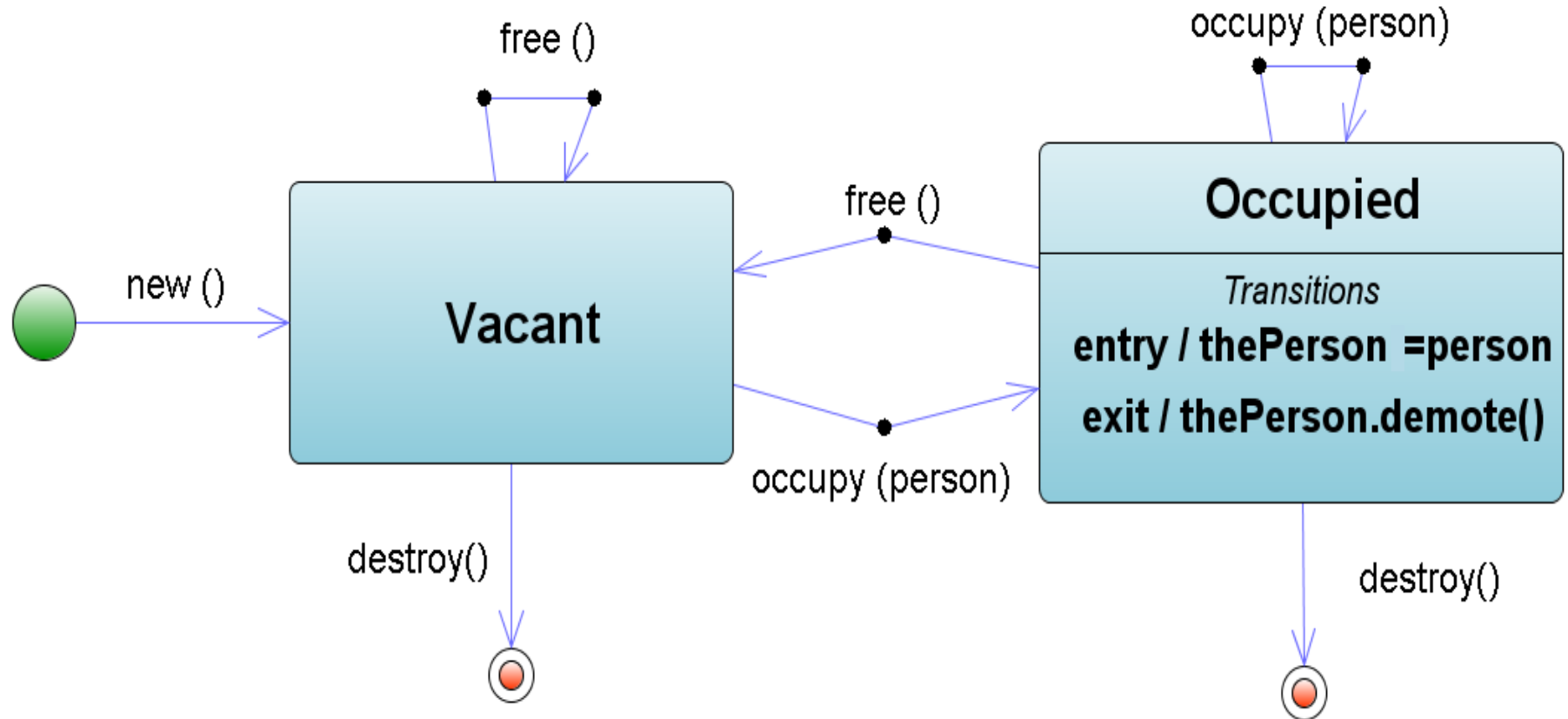
Пример ИС ОК: назначение сотрудника на должность



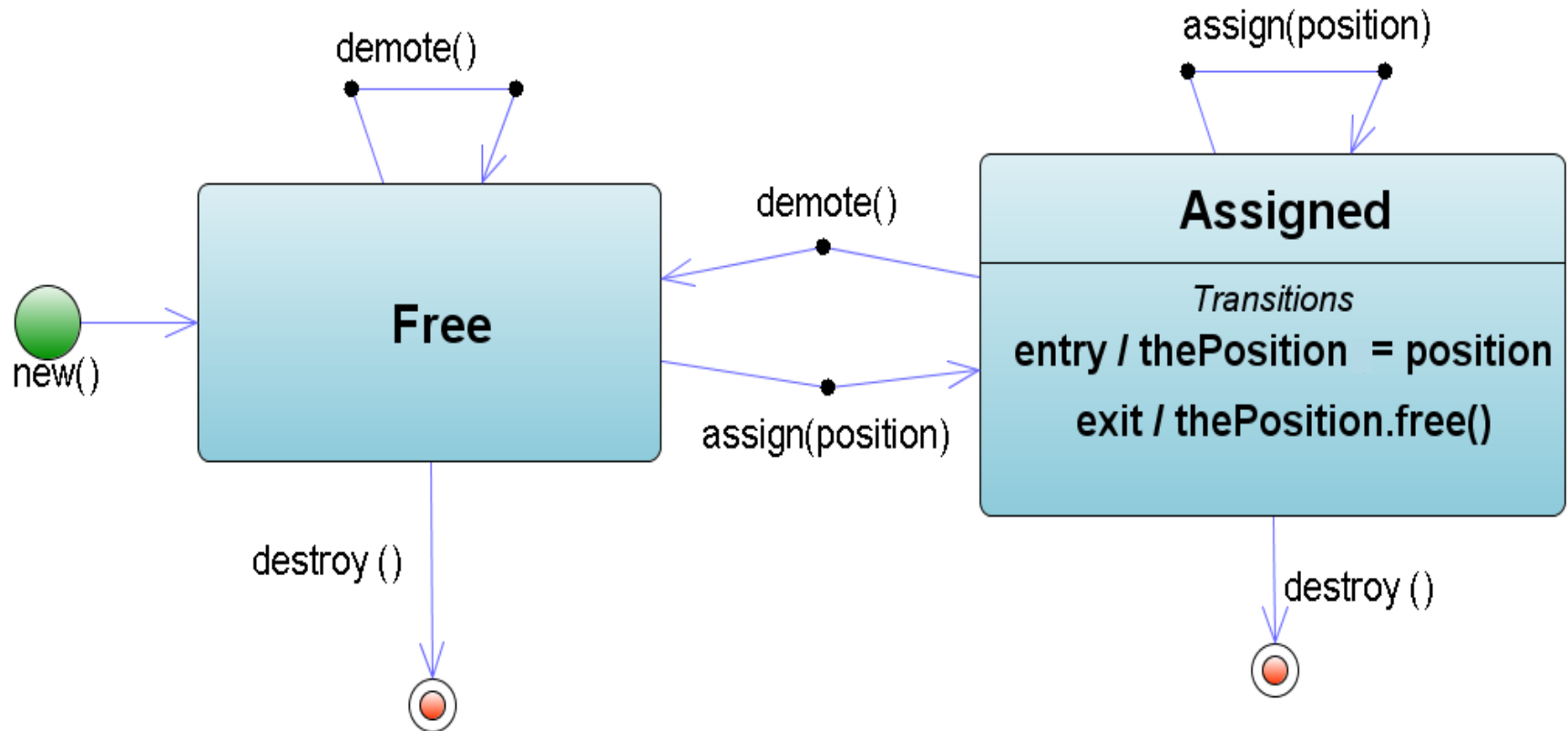
Изменение состояний при выполнении операции move



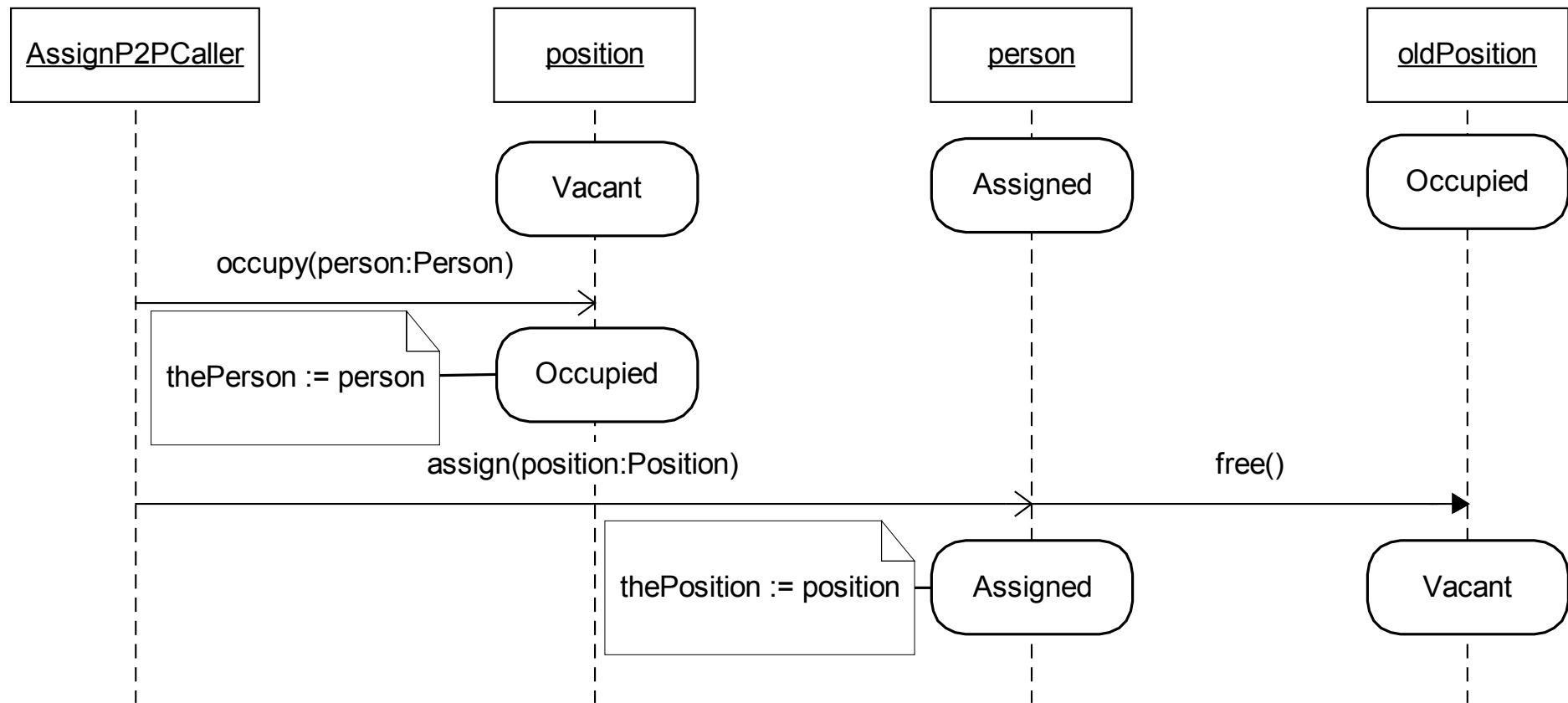
Автомат класса Position



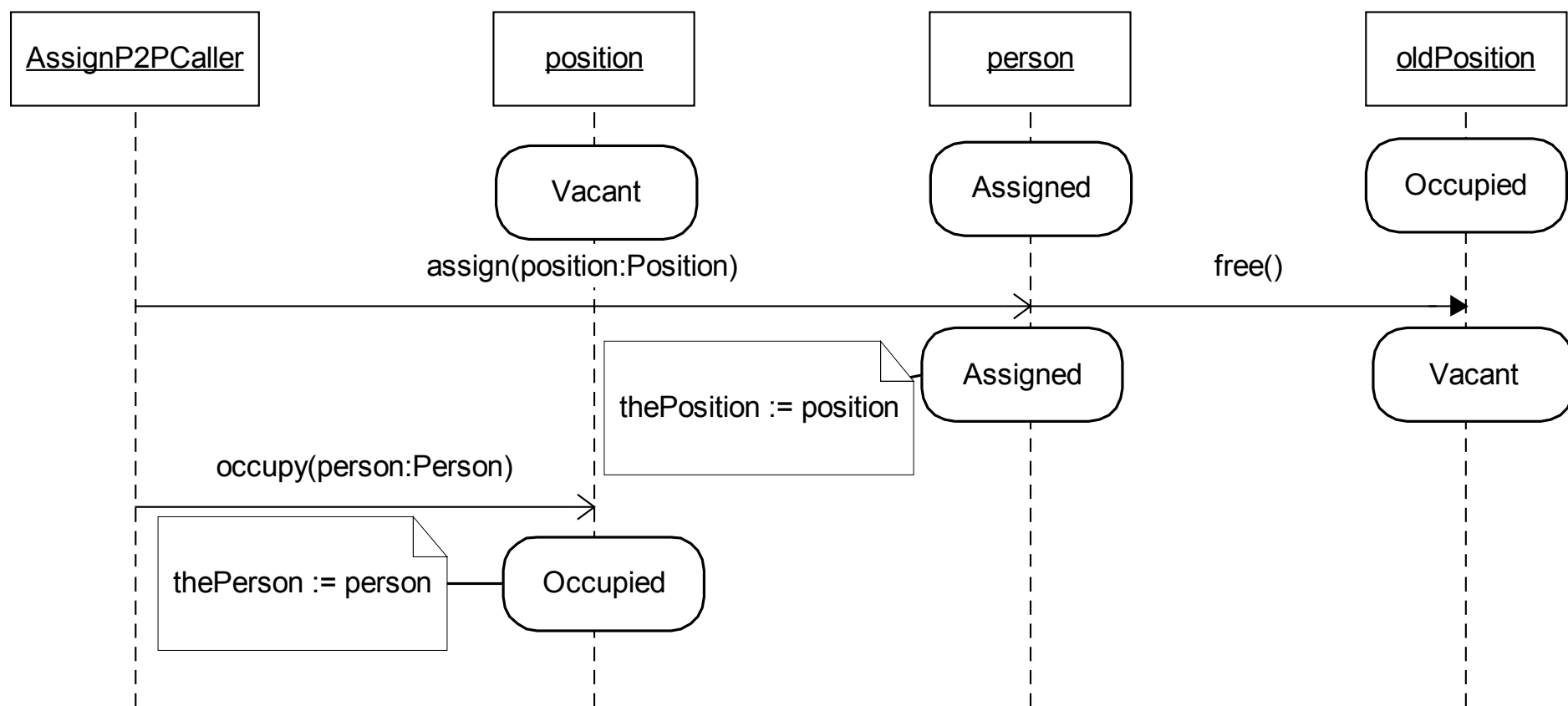
Автомат класса Person



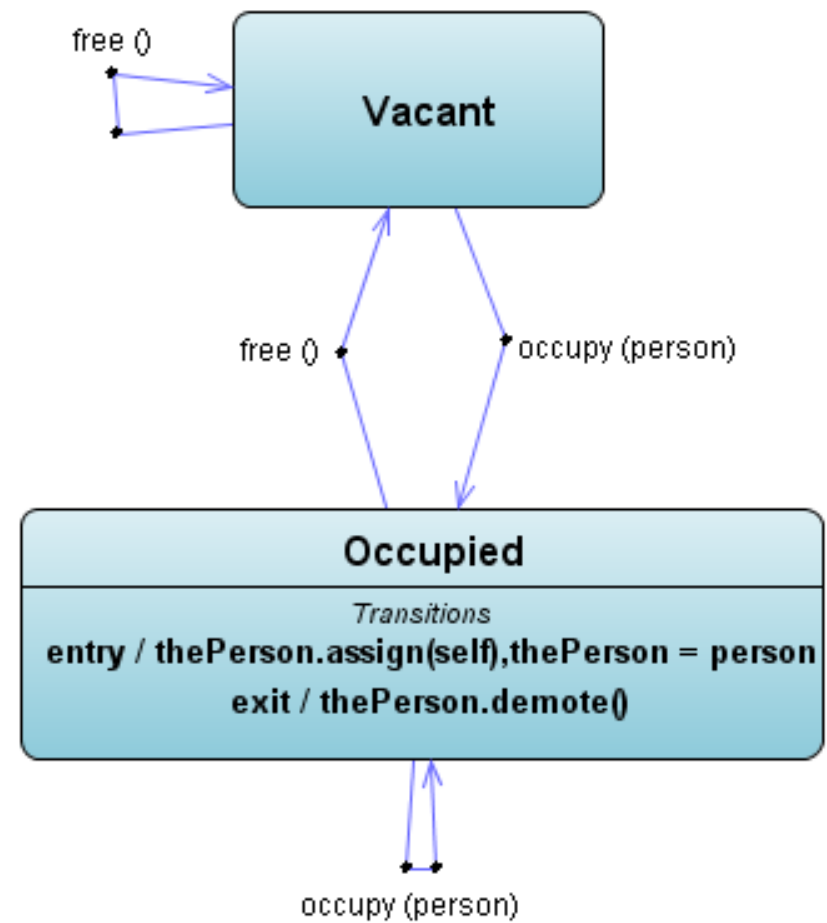
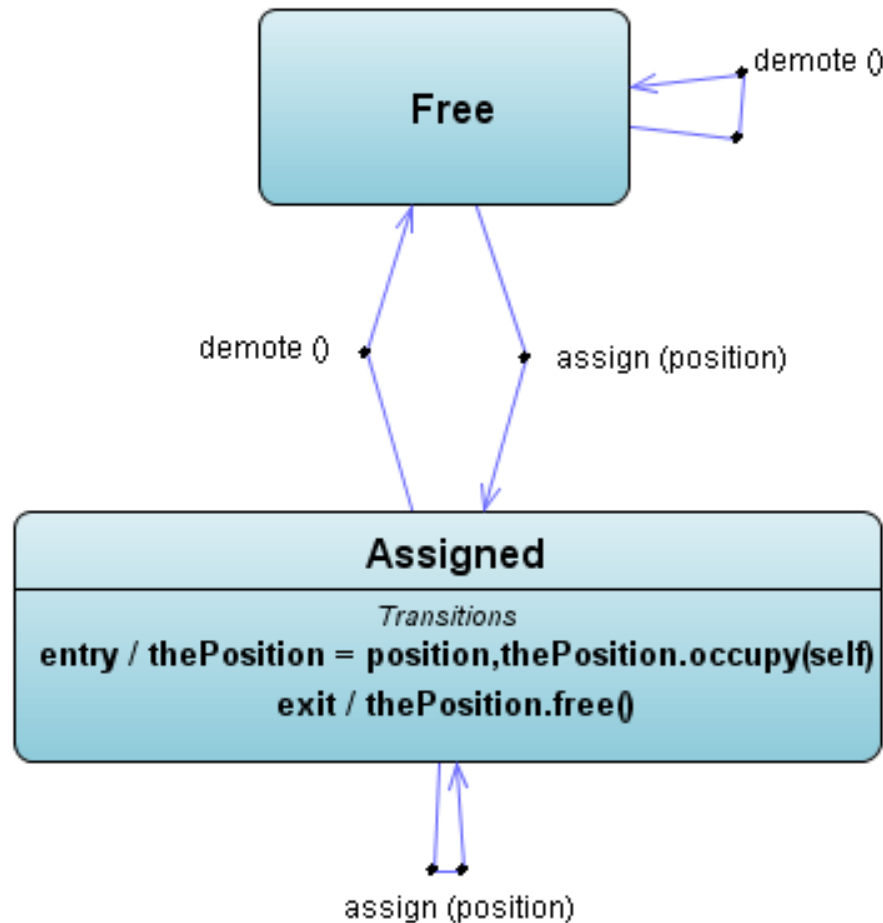
Первый пример сценария



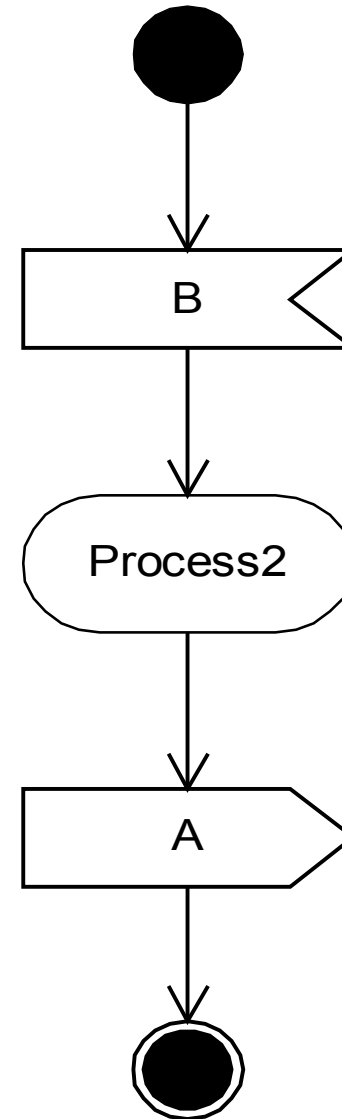
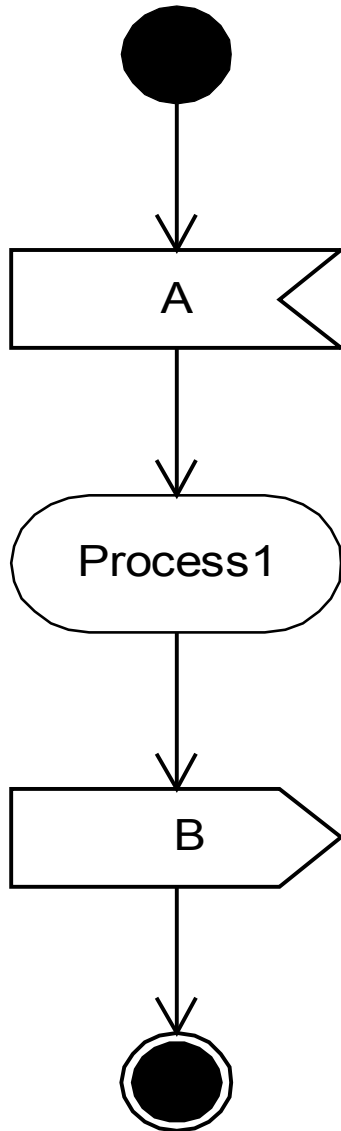
Второй пример сценария



Ошибка 1: взаимная рекурсия



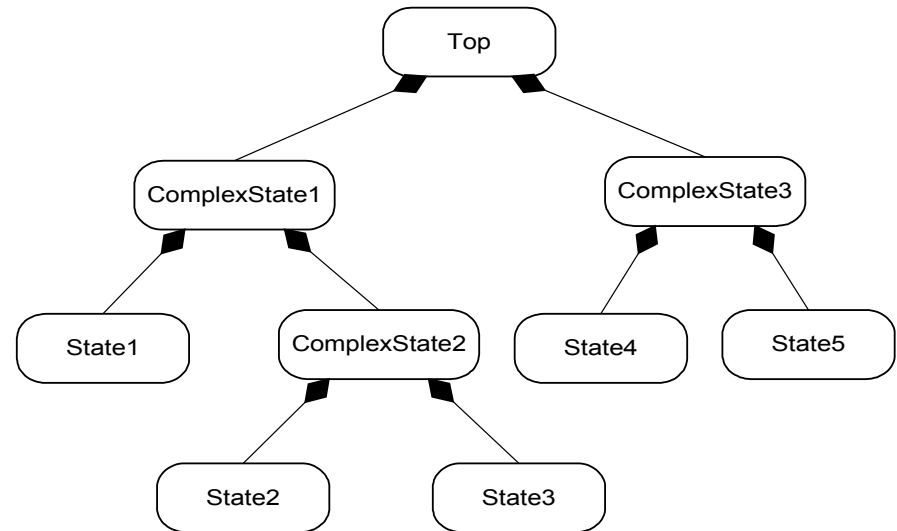
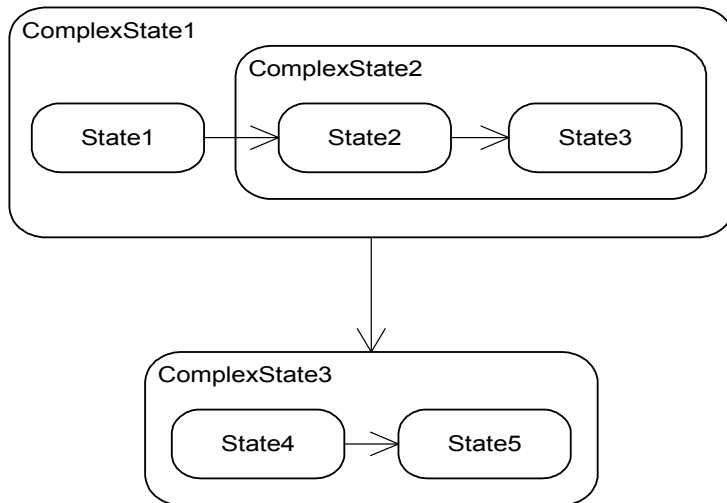
Ошибка 2: взаимная блокировка



Параллельные составные состояния и составные переходы

- **Составное последовательное состояние: множество состояний вложенных автоматов – путь в дереве вложенности**
- **Составное параллельное состояние: гиперпуть в дереве И/ИЛИ**
- **Составной переход: переход из составного состояния в и/или из составного состояния**
- **Нотация: развилки / слияния**

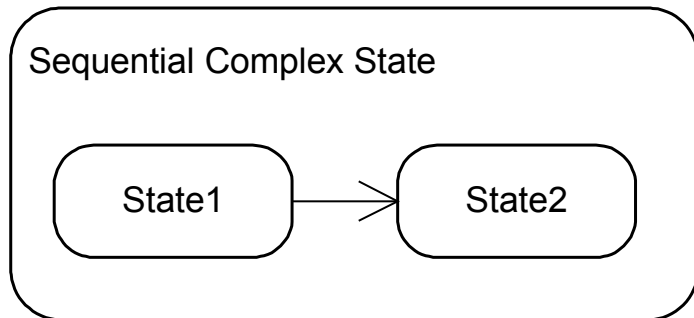
Дерево составных последовательных состояний



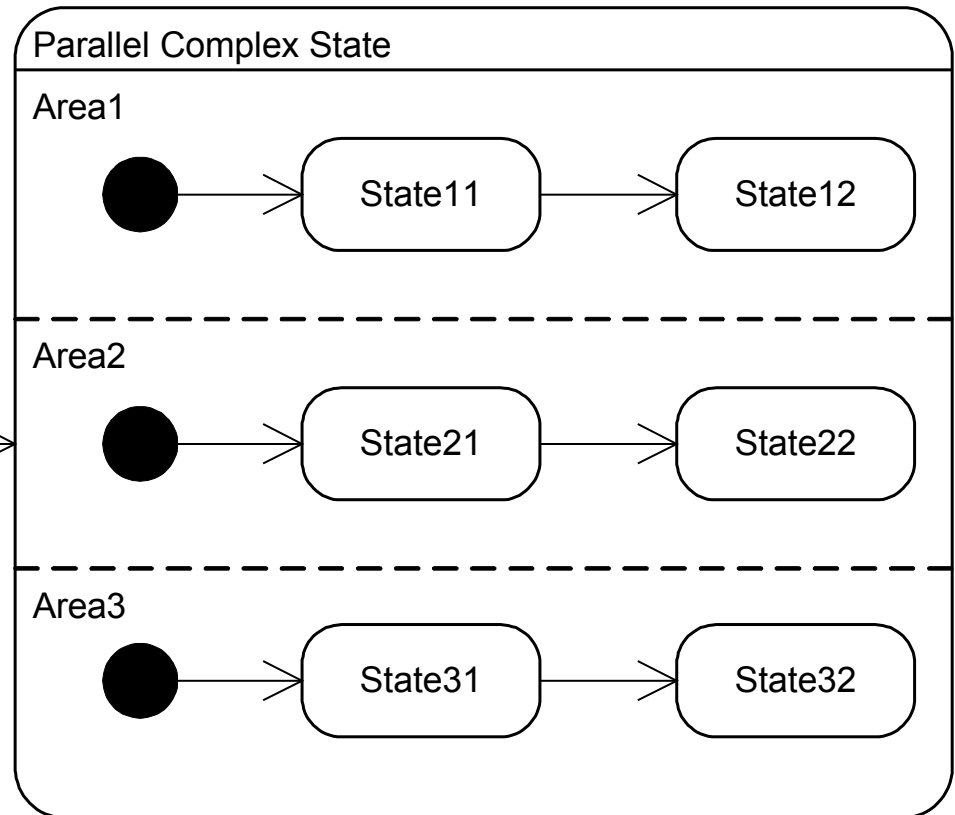
Параллельное (ортогональное) составное состояние

Параллельное составное состояние

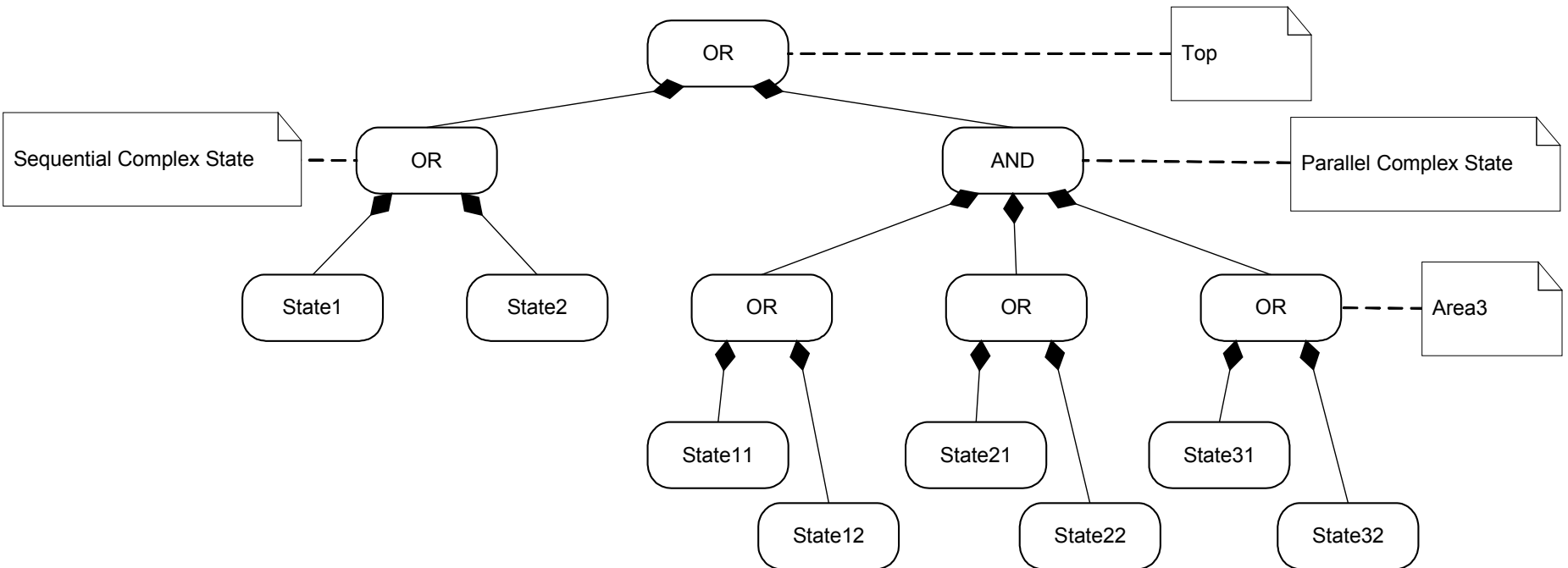
Последовательное
составное
состояние



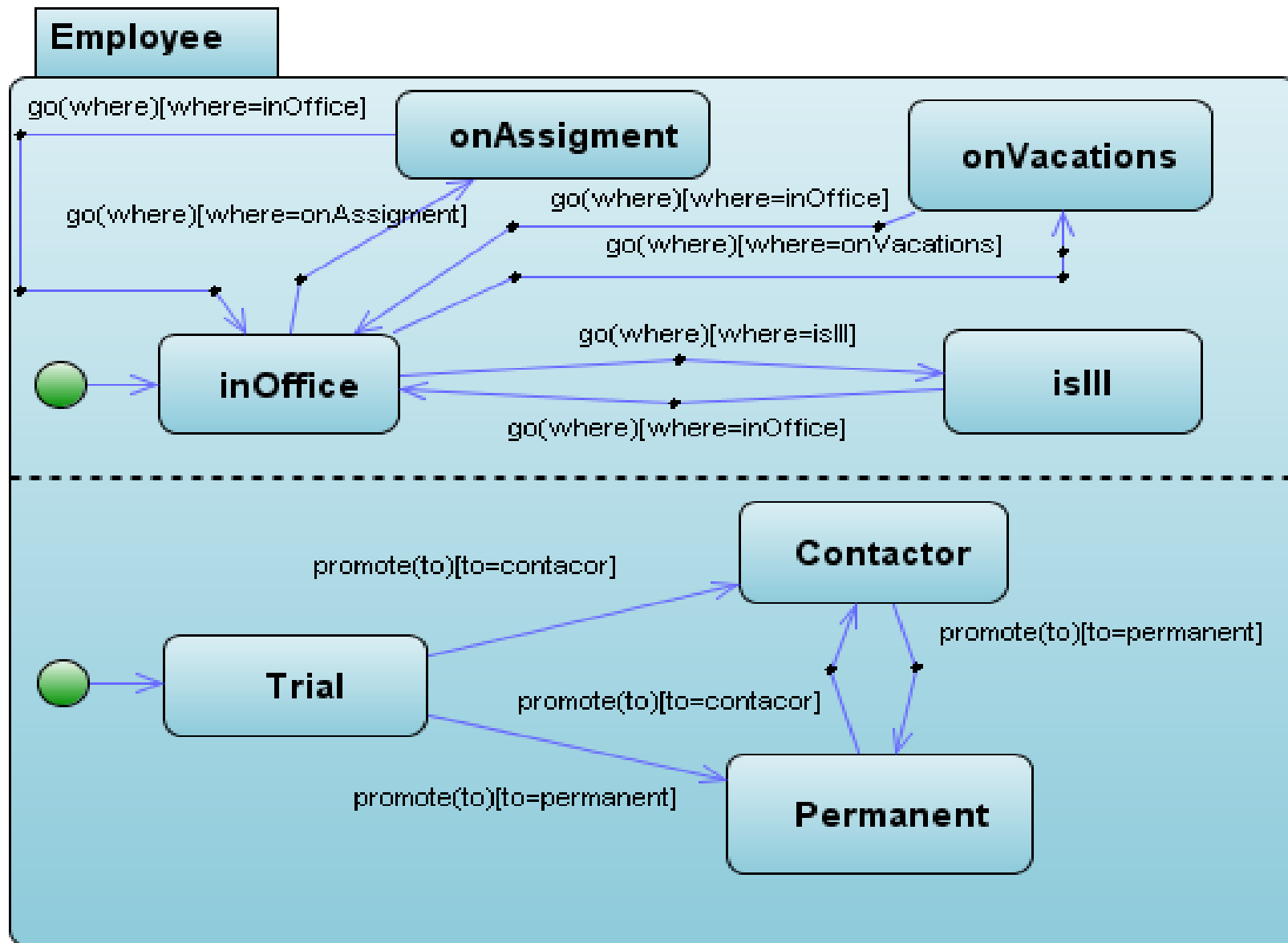
Область
параллельного
составного
состояния



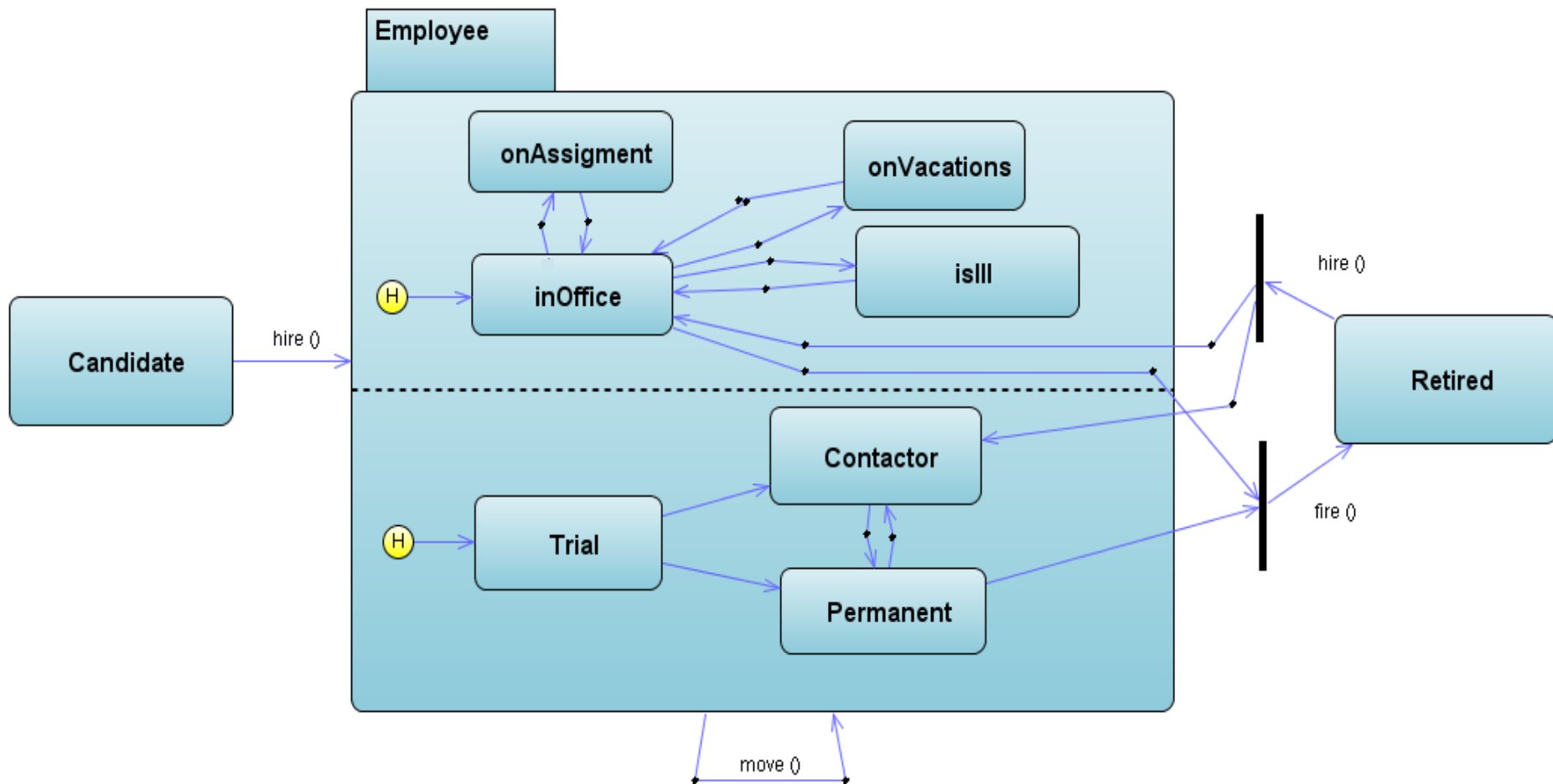
Дерево И/ИЛИ



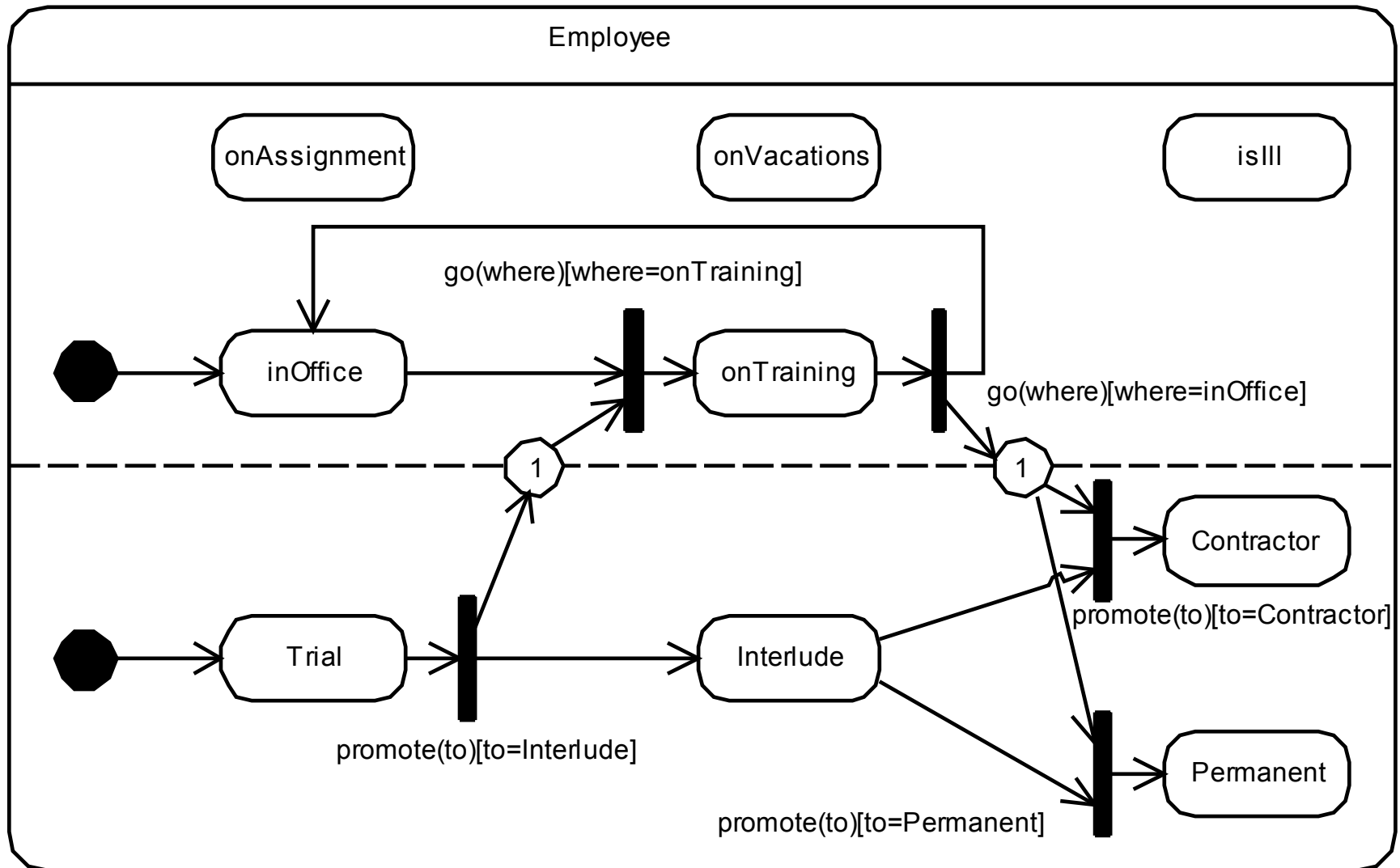
Пример ИС ОК



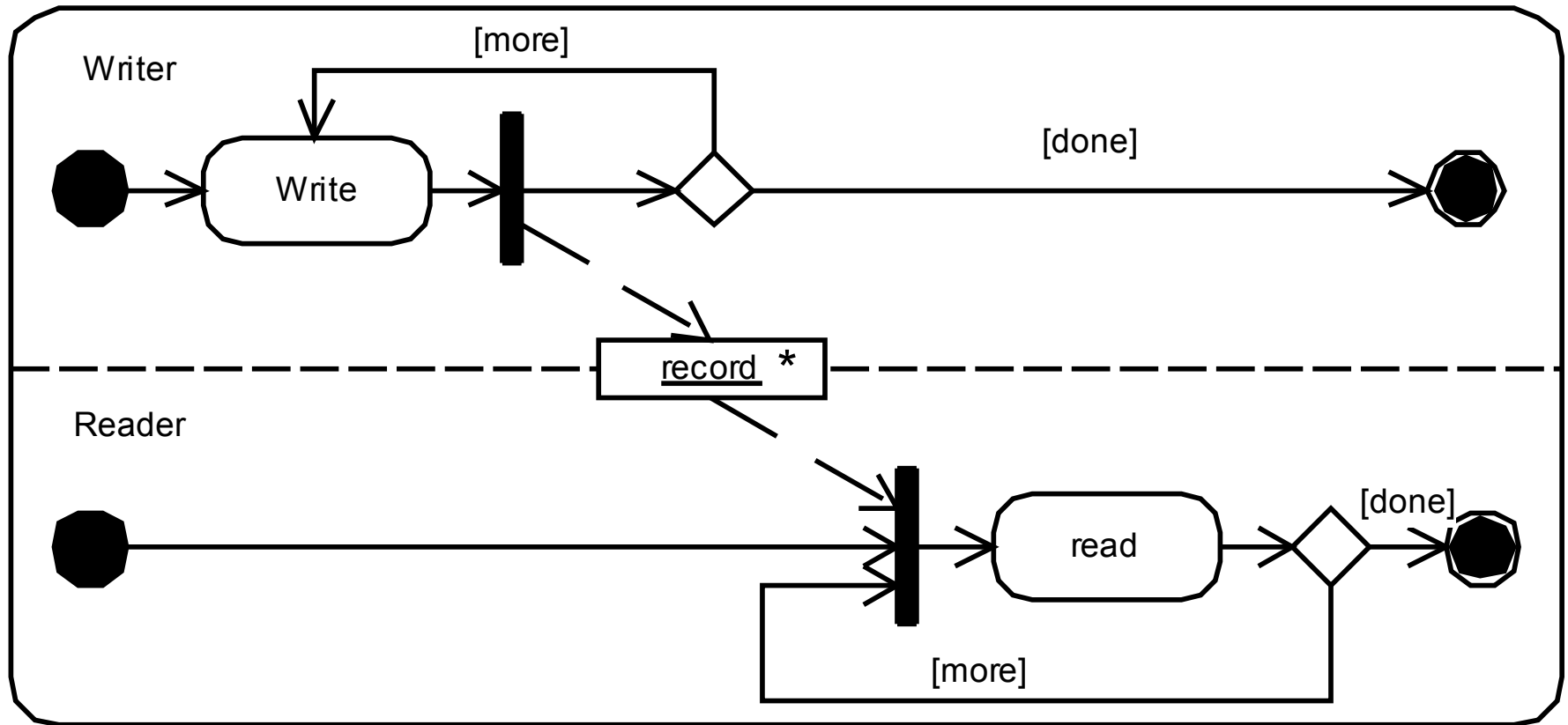
Составной переход



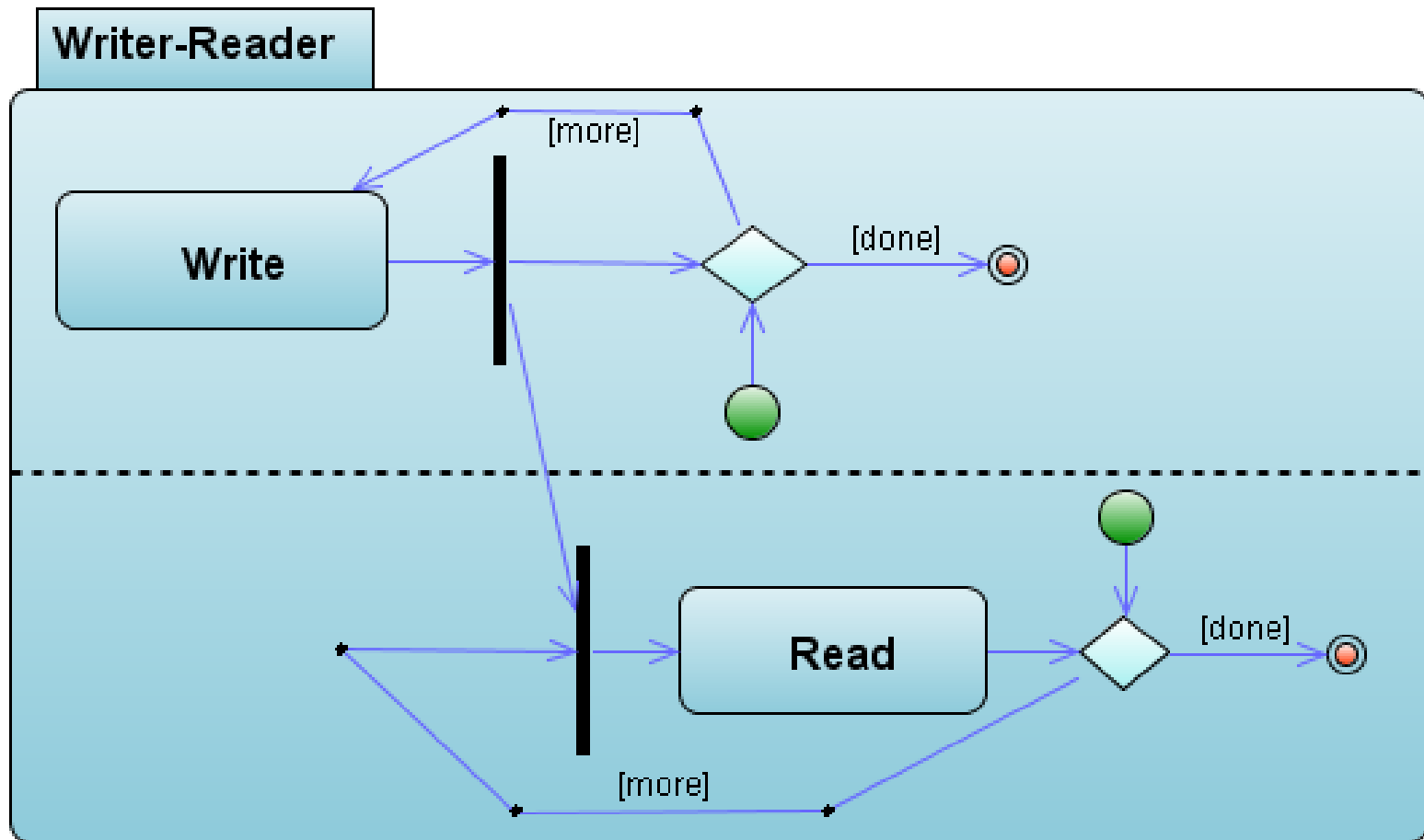
Синхронизирующие состояния (1.x)



Синхронизирующий объект в состоянии (1.x, 2.0👉)



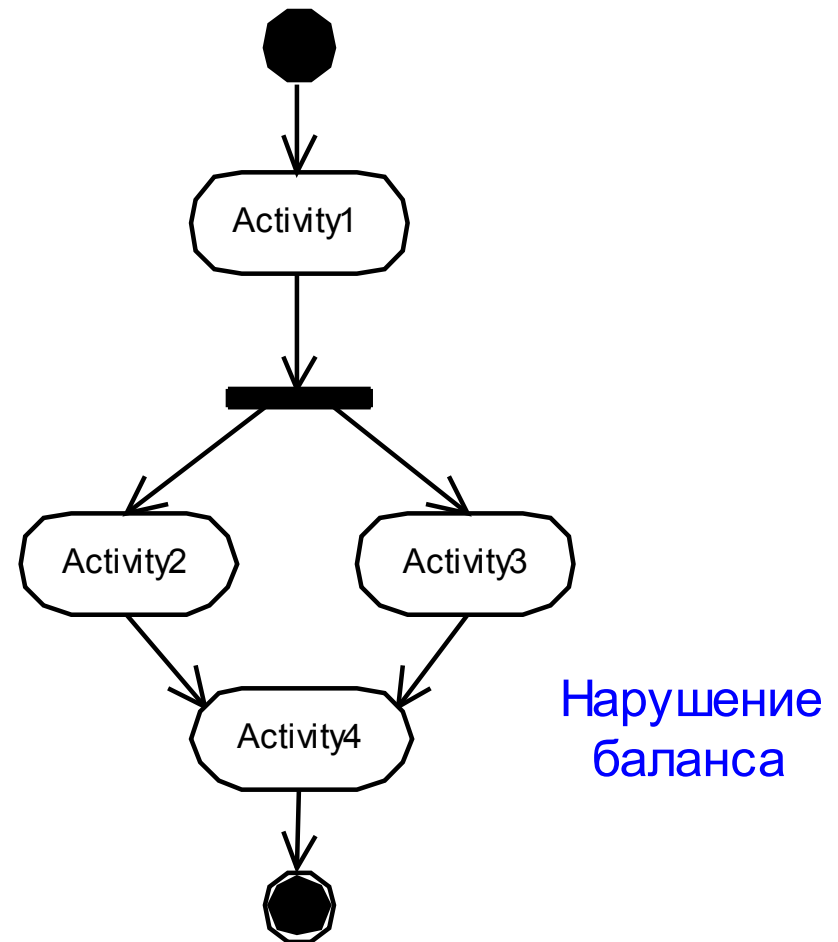
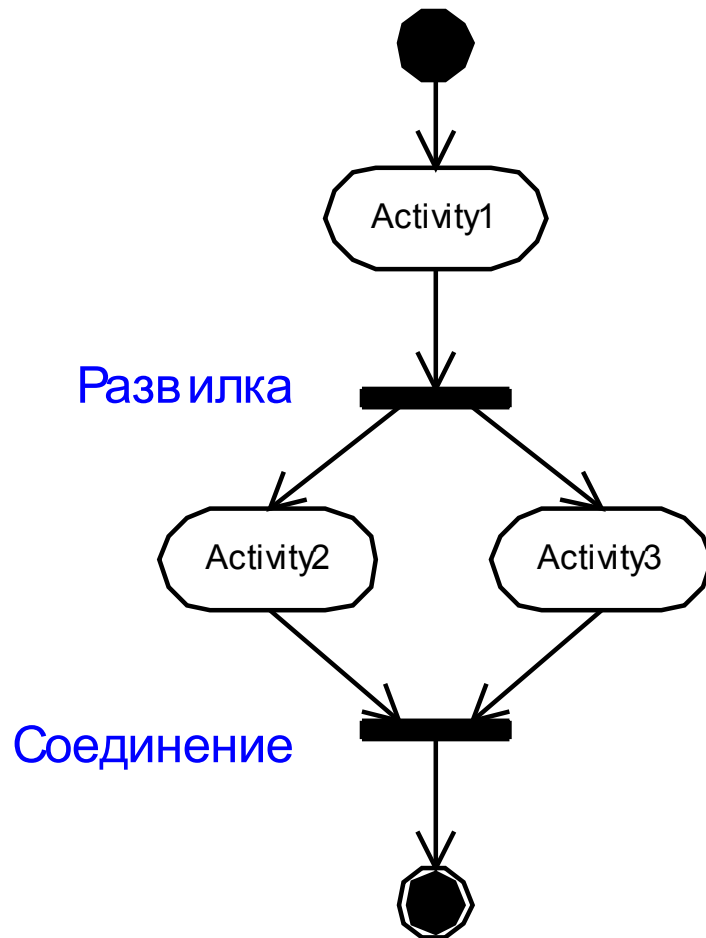
Писатель – читатель



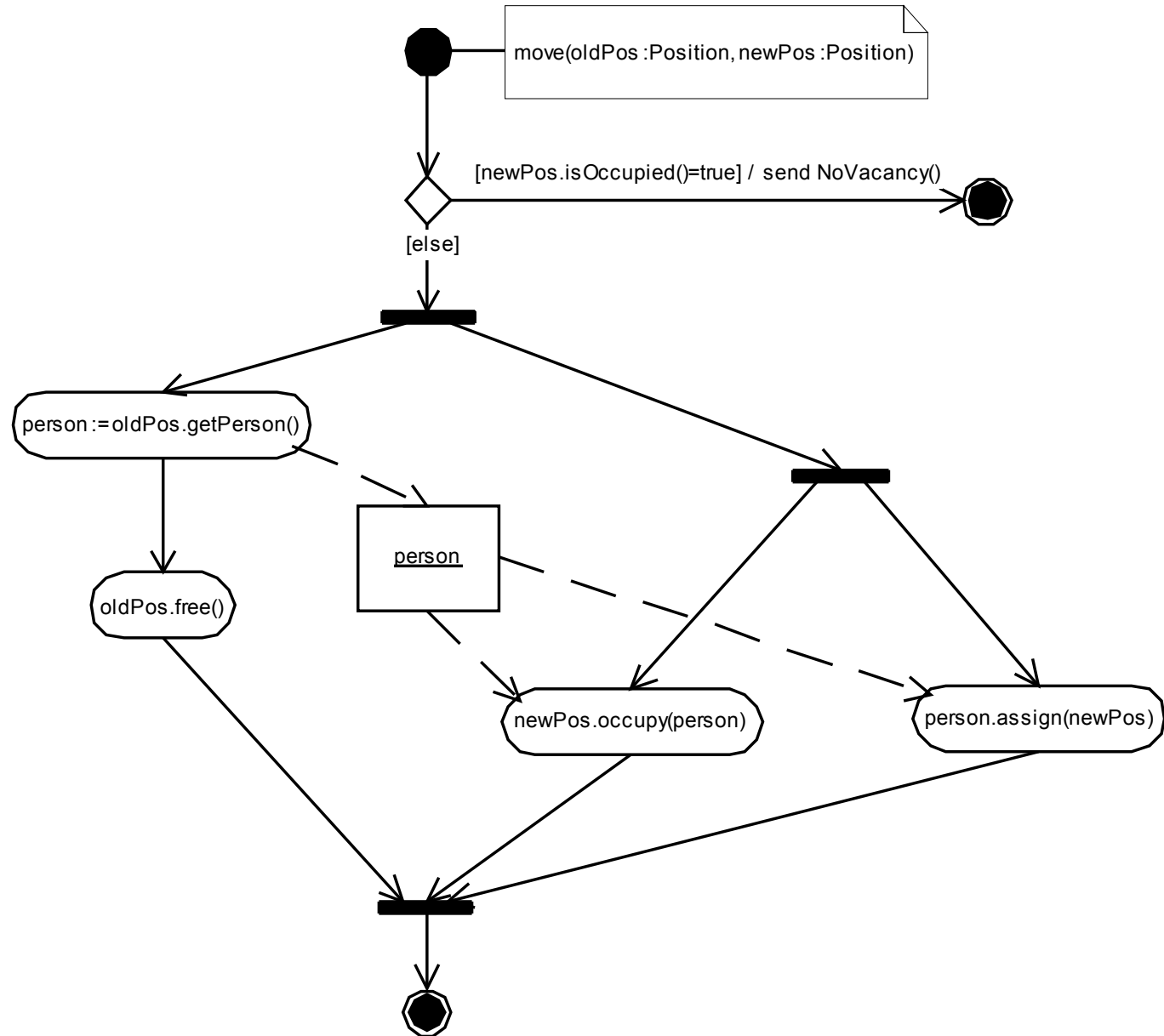
Развилки и соединения

- **Квазипараллельные потоки управления**
- **Развилка (fork)**
 - **один вход, 2..* выходов**
- **Соединение (join)**
 - **2..* входов, один выход**
- **Синхронизация в точке слияния**
- **Баланс развилок / соединений**

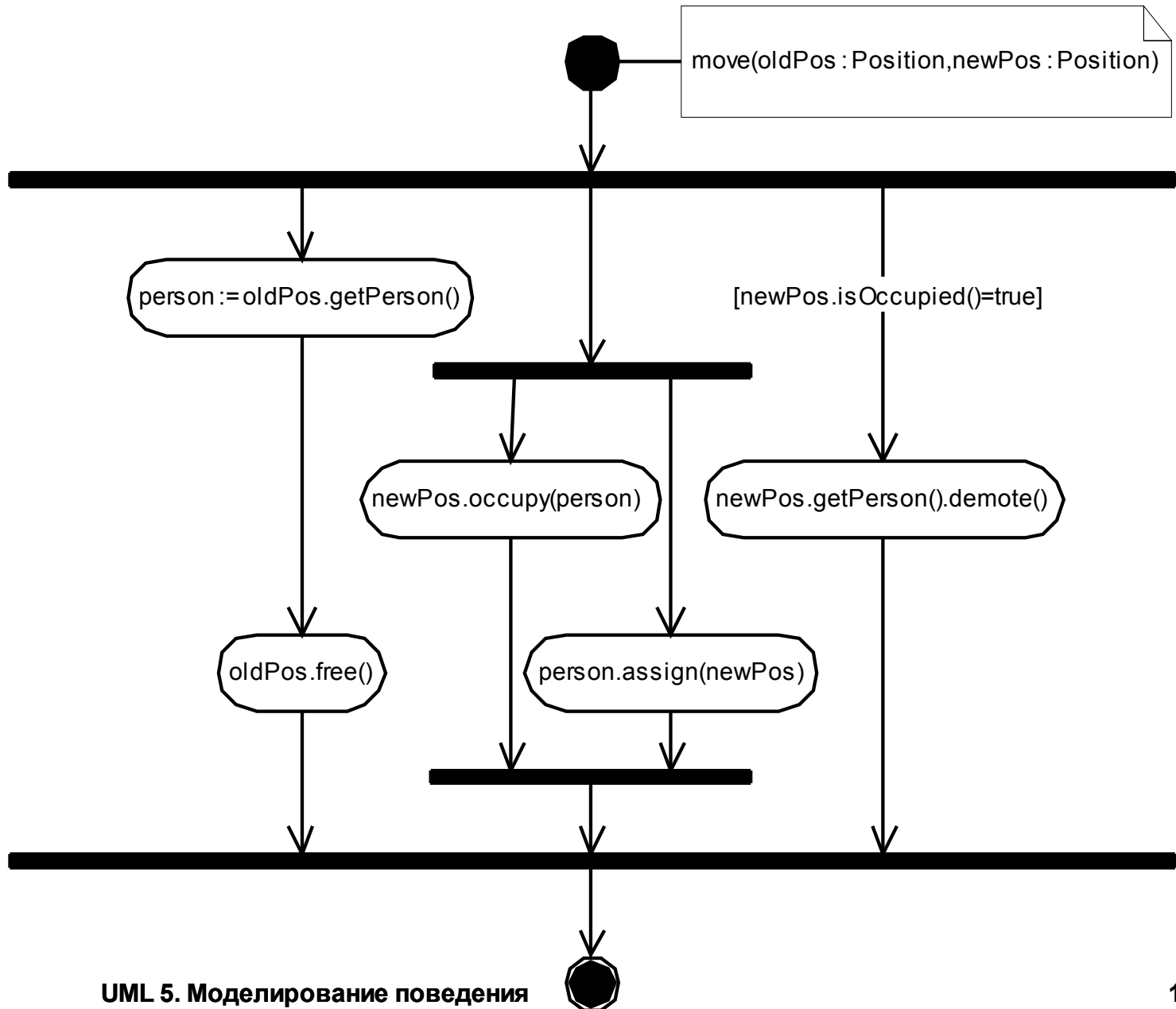
Баланс развилки и соединений



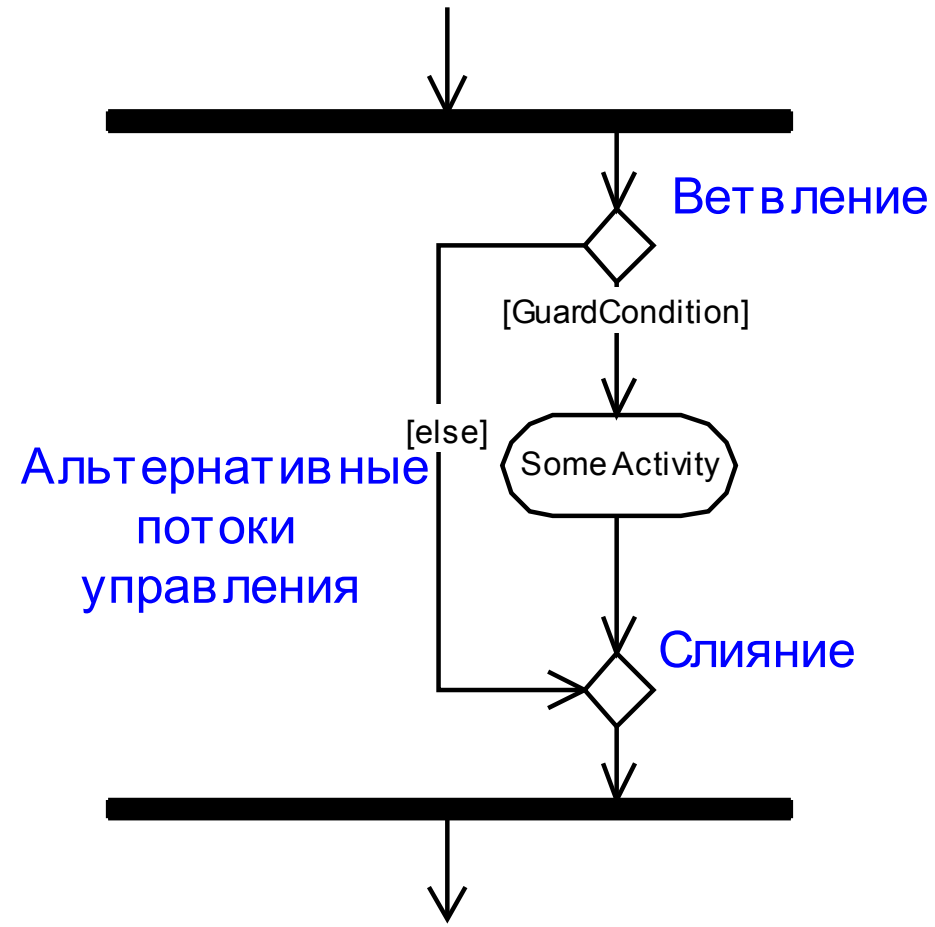
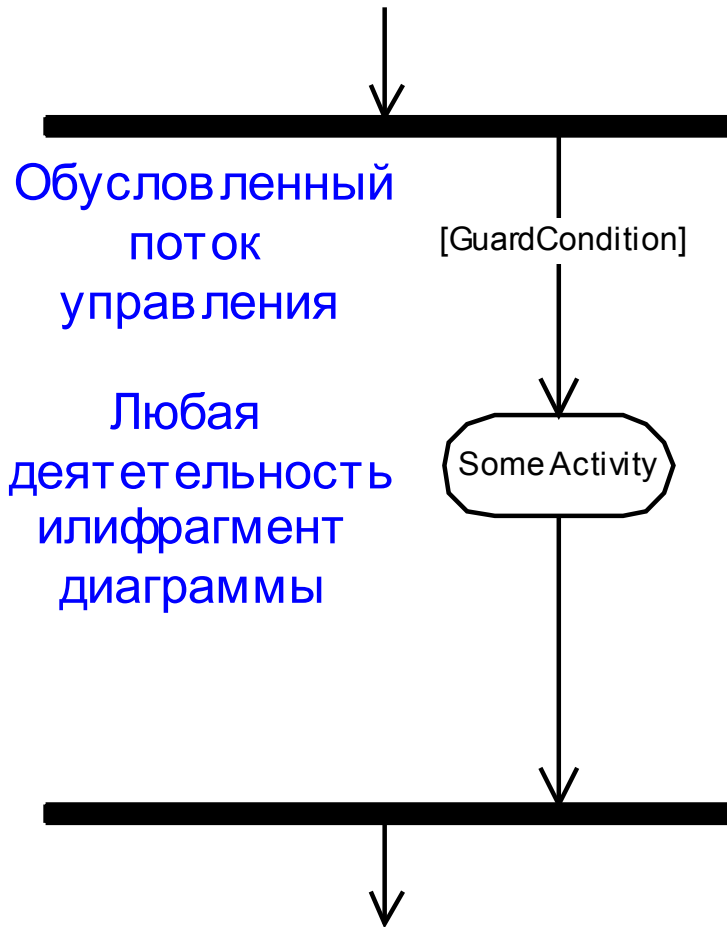
Пример ИС ОК: Перевод сотрудника



Обусловленный поток управления



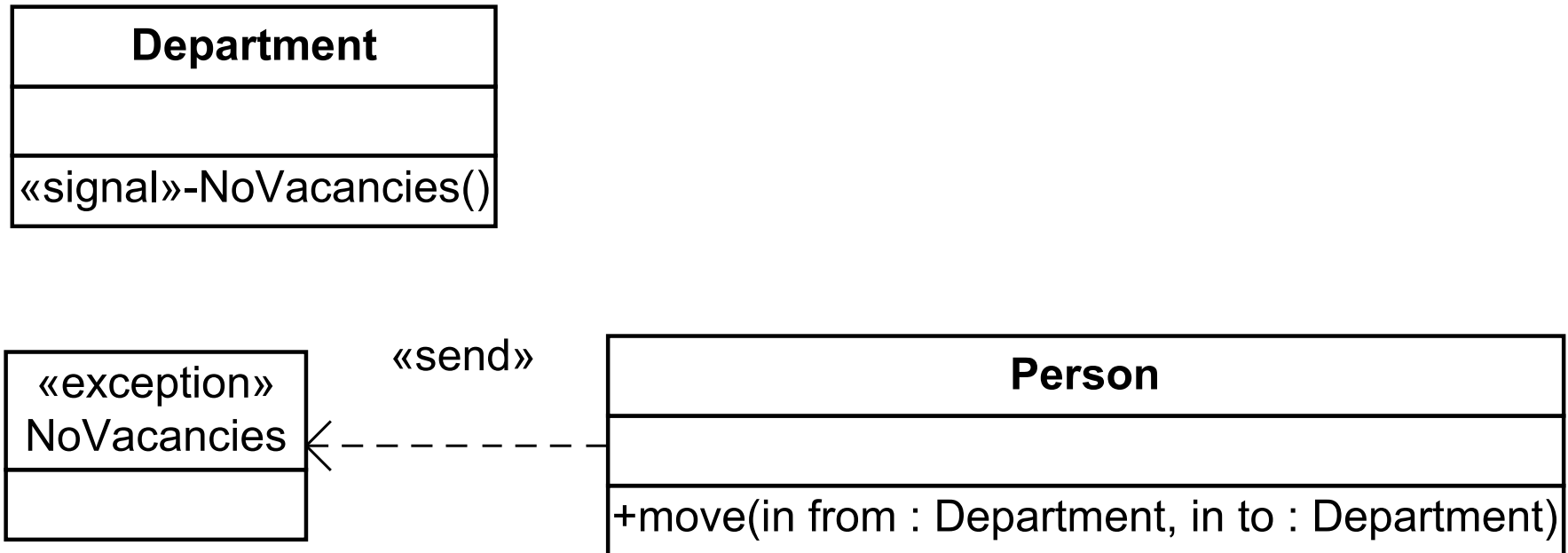
Выражение обусловленного потока управления через ветвление



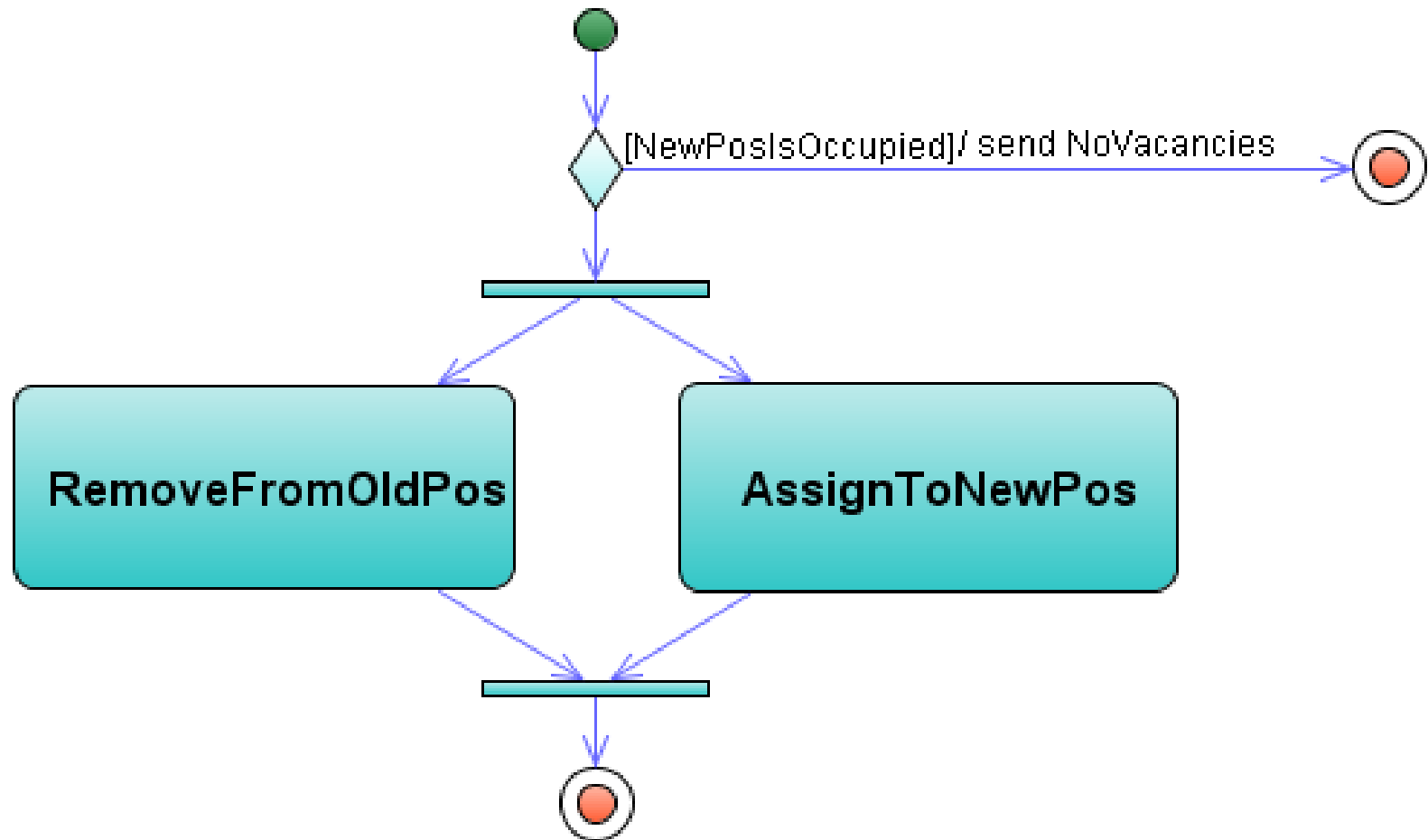
Сигналы

- Сигнал – это именованный объект, который создается другим объектом (отправителем) и перехватывается третьим объектом (получателем)
- Исключения (exception) – в UML частный случай сигналов
- Сигнал, как объект, может иметь атрибуты (т.е. параметры)
- На диаграмме классов сигнал связывается с отправителем зависимостью со стереотипом «send» и связывается с получателем объявлением операции со стереотипом «signal»

Пример ИС ОК: обработка исключительных ситуаций



Пример ИС ОК: обработка исключительных ситуаций



Поток (thread) и процесс (process)

- Описываются как активный класс со стереотипом «process» или «thread»
- Нотация : жирный прямоугольник (1.x) или двойные линии на боковых сторонах (2.0)
- Экземпляр такого класса моделирует отдельный поток управления
- Внутри м.б. нарисована кооперация взаимодействующих объектов

Выводы

- Конечные автоматы — базовая алгоритмическая техника моделирования поведения
- Диаграмма состояний — вариант конечного автомата с различными дополнениями
- Диаграмма деятельности (= блок-схема) — частный случай диаграммы состояний с различными дополнениями
- Диаграммы кооперации и последовательности (диаграммы взаимодействия) семантически эквиваленты и являются протоколами выполнения (примерами)
- Параллелизм на уровне программы моделируется с помощью взаимодействующих машин состояний
- Параллелизм на уровне операционной системы моделируется активными объектами