ГЛАВА 2
МОДЕЛИРОВАНИЕ ИСПОЛЬЗОВАНИЯ

2.1. ЗНАЧЕНИЕ МОДЕЛИРОВАНИЯ ИСПОЛЬЗОВАНИЯ

Для большинства средств UML нетрудно отыскать аналоги среди широко используемых практических методов конструирования программных систем. И это не удивительно, ведь именно эти методы были унифицированы посредством UML. А вот для диаграмм использования известный аналог указать труднее. Мы попытаемся объяснить прагматику моделирования использования на конкретном примере.

2.1.1. Сквозной пример

- В остальных частях учебного пособия рассматривается один сквозной Пример моделирования сравнительно приложения — информационной системы отдела кадров. Выбор примера обусловлен следующими соображениями. Во-первых, предметная область до некоторой знакома всем. Таким образом, суть задачи заранее ясна, и можно сосредоточить внимание на тонкостях применения UML, а не на объяснении особенностей предметной области.

- Во-вторых, информационная система отдела кадров — это типичное офисное приложение из самого распространенного класса систем автоматизации делопроизводства. UML как нельзя лучше подходит для моделирования именно таких систем и все средства языка можно проиллюстрировать естественным образом.
- В-третьих, авторам случалось разрабатывать системы на самом деле, а не только в книге.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Информационная система «Отдел кадров» (сокращенно ИС ОК) предназначена для ввода, хранения и обработки информации о сотрудниках и движении кадров. Система должна обеспечивать выполнение следующих основных функций.

1. Прием, перевод и увольнение сотрудников.

2. Создание и ликвидация подразделений.

2. Создание вакансий и сокращение должностей.

Конечно, техническое задание из одного абзаца текста и трех нумерованных пунктов — это не более чем учебный пример[4]. Однако даже на этом примере видны многие характерные "особенности" подобных документов, которые, увы, слишком часто встречаются в реальной жизни. С одной стороны, что-то написано, а с другой стороны не очень понятно, что делать дальше. Безо всяких объяснений заказчик использует термины свой предметной области — разработчик должен их знать и понимать. Требований к реализации нет вовсе. Функции не упорядочены по приоритетам: не ясно, что является критически важным, а чем можно поступиться в случае необходимости.

2.1.2. Преимущества моделирования использования

Отвлечемся пока от технических деталей нотации диаграмм использования и рассмотрим, что предлагается делать на первом шаге моделирования использования.

Перечислим те преимущества, которые дает этот подход по сравнению с другими.

Простые утверждения. Моделирование использования фактически позволяет переписать исходное техническое задание (или просто записать, если никакой исходной формулировки требований не было) в строгой и формальной, но, в тоже время, очень простой и наглядной графической форме, как совокупность простых утверждений относительно того, что делает система для пользователей. Конечно, использование такой формы не гарантирует от ошибок (вряд ли гарантия от ошибок вообще возможна), но благодаря простоте и наглядности формы их легче заметить.

Абстрагирование от реализации. Моделирование использования предполагает формулирование требований к системе абсолютно независимо от ее реализации. Другими словами, представление использования описывает только, **что** делает система (но не **как** это делается и не **зачем** это нужно делать). Заметим, что другие подходы, используя на первых шагах термины и понятия реализации (структура программы, структура данных, структура взаимодействующих объектов) накладывают невольные ограничения на реализацию, которые не вытекают из существа задачи, а значит, могут служить источником неэффективности и ошибок.

Декларативное описание. Каждый вариант использования описывает (а вернее сказать, именует) некоторое множество последовательностей действий, доставляющих значимый для пользователя результат. Однако никакого императивного описания представление использования не содержит, в модели нет указаний на то, какой вариант использования должен выполняться раньше, а какой позже, то есть, нет описания алгоритма, а значит, нет алгоритмических ошибок.

Выявление границ. Представление использования определяет границы системы и постулирует существование во внешнем мире использующих ее агентов (действующих лиц). Описание системы в виде черного ящика с определенными интерфейсами кажется очень похожим на представление использования, но здесь есть важное различие, которое часто упускается из вида. Если ограничиться только описанием интерфейсов, то очень легко допустить ошибки следующего типа: предусмотреть интерфейс, который не нужен, потому что им никто не пользуется. Или, аналогично, забыть интерфейс, который необходим определенной категории пользователей. На диаграмме использования одинокие и покинутые действующие лица и варианты использования обнаруживаются с первого взгляда.

Наш вывод таков: **моделирование использования безопаснее и надежнее альтернативных методов**, то есть при прочих равных условиях позволяет совершить меньше грубых проектных ошибок на ранних стадиях проектирования. В этом заключается основное преимущество данного метода.

2.2. ДИАГРАММЫ ИСПОЛЬЗОВАНИЯ

Диаграммы использования были предложены Иваром Якобсоном в их нынешней графической форме еще в 1986 году. Диаграммы использования являются, безусловно, самым стабильным элементом UML — они не менялись уже двадцать лет с лишним, фактически, приняли законченную форму задолго до появления языка. Одновременно эти диаграммы имеют самую простую нотацию: всего два основных типа сущностей (действующие лица и варианты использования) и три типа отношений (зависимости, ассоциации, обобщения)!

2.2.1. Действующие лица

Вопрос о выделении (или идентификации) действующих лиц при составлении модели — один из самых болезненных. Неудачный выбор действующих лиц может отрицательно повлиять на всю модель в целом. Здесь легко впасть в крайность: объявить, что имеется одно действующее лицо (внешний мир), взаимодействующее со всеми вариантами использования или, наоборот, придумать искусственных действующих лиц для каждого варианта использования. Оба экстремальных варианта являются, по существу, моделью черного ящика и сводят к нулю преимущества моделирования использования,

рассмотренные в предыдущем разделе. Формального метода идентификации действующих лиц не существует. Здесь мы перечислим некоторые приемы, которые полезно иметь в виду при выделении действующих лиц и покажем применение этих приемов на нашем примере информационной системы отдела кадров. Для начала укажем более детальное определение действующего лица.

С синтаксической точки зрения ***действующее лицо*** *— это стереотип классификатора, который обозначается специальным значком. Для действующего лица указывается только имя, идентифицирующее его в системе. Семантически действующее лицо — это множество логически взаимосвязанных ролей.*

С прагматической точки зрения главным является то, что действующие лица находятся **вне** проектируемой системы (или рассматриваемой части системы).

В типовых случаях различные действующие лица назначаются для категорий пользователей (если их удается выделить естественным образом), внешних программных и аппаратных средств (если система взаимодействует с таковыми).

Рассмотрим наш пример с информационной системой отдела кадров. Выделение категорий пользователей происходит, как правило, неформально: из соображений здравого смысла и собственного опыта. Тем не менее, несколько советов мы можем дать. Имеет смысл отнести пользователей к разным категориям, если наблюдаются следующие признаки:

- пользователи участвуют в разных (независимых) бизнес-процессах;

- пользователи имеют различные права на выполнение действий и доступ к информации;

- пользователи взаимодействуют с системой в разных режимах: от случая к случаю, регулярно, постоянно.

Опираясь на собственные советы, применительно к нашему примеру мы в первом приближении выделяем две категории пользователей:

- менеджер персонала, который работает с конкретными людьми;

- менеджер штатного расписания, который работает с абстрактными должностями и подразделениями.

Бизнес-процесс пользователя первой категории включает в себя не только работу с приложением, но и беседы с конкретными людьми, интервью и тому подобное, чем явно отличается от других бизнес-процессов предприятия.

Пользователи второй категории, очевидно, должны иметь специальные права доступа, поскольку вряд ли допустимо, чтобы кто угодно мог создавать и уничтожать подразделения на предприятии.

На рис. 2.1 мы начинаем формировать представление использования информационной системы отдела кадров. Менеджер персонала имеет имя `Personnel Manager`, а менеджер штатного расписания — `Staff Manager`, в соответствии с используемой дисциплиной имен.

Рис. 2.1. Действующие лица ИС ОК

Для UML пока что нет достаточно устоявшейся дисциплины имен, но некоторый набор рекомендаций можно найти в литературе. Мы, по возможности, следуем этим рекомендациям. В частности, в качестве имен действующих лиц рекомендуется использовать существительное (возможно с определяющим словом), а в качестве имен вариантов использования — глагол (возможно, с дополнением). Эти правила основаны на семантике моделирования использования.

2.2.2. Варианты использования

Выделение вариантов использования — ключ ко всему дальнейшему моделированию. На этом этапе определяется функциональность системы, то есть, **что** она должна делать.

Нотация для варианта использования очень скудная — это просто имя, помещенное в овал (или помещенное под овалом — такой вариант тоже допустим). Другими словами, функции, выполняемые системой, на уровне моделирования использования никак не раскрываются — им только даются имена.

Семантически ***вариант использования*** *(use case) — это описание множества возможных последовательностей действий (событий), приводящих к значимому для действующего лица результату.*

Каждая конкретная последовательность действий называется ***сценарием****.*

В нашем примере простой анализ текста технического задания выявляет семь вариантов использования:

- прием сотрудника;
- перевод сотрудника;
- увольнение сотрудника;
- создание подразделения;
- ликвидация подразделения;
- создание вакансии;
- сокращение должности.

Опираясь на знание предметной области, получим набор вариантов использования, представленный на рис. 2.2.

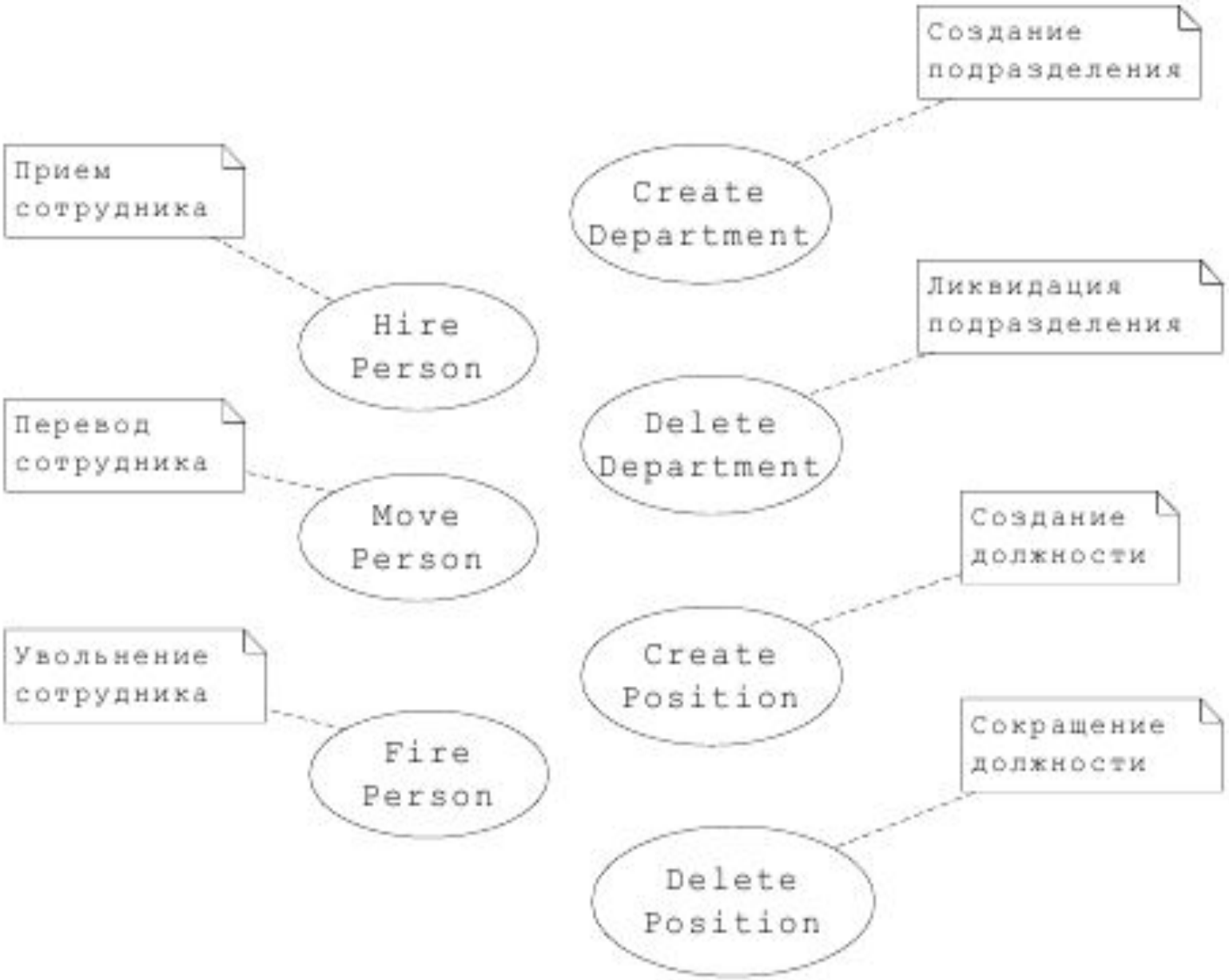


Рис. 2.2. Варианты использования ИС ОК

2.2.3. Примечания

Третьим типом сущности, применяемым на диаграмме использования, является примечание. Заметим, что примечания являются очень важным средством UML, значение которого часто недооценивается начинающими пользователями. **Примечания можно и нужно употреблять на всех типах диаграмм**, а не только на диаграммах использования. UML является унифицированным, но никак не универсальным языком — при моделировании проектировщик часто может сказать о моделируемой системе больше, чем это позволяет сделать строгая, но ограниченная нотация UML. В таких случаях наиболее подходящим средством для внесения в модель дополнительной информации является примечание.

В отличие от большинства языков программирования примечания в UML синтаксически оформлены с помощью специальной нотации и выступают на тех же правах, что и остальные сущности. А именно, примечание имеет свою графическую нотацию — прямоугольник с загнутым уголком ("собачье ухо"), в котором находится текст примечания. Примечания могут находиться в отношении соответствия с другими сущностями — эти отношения изображаются пунктирной линией без стрелок. Если пунктирная линия отсутствует, то примечание относится ко всей диаграмме.

Примечания содержат текст, который вводит пользователь — создатель модели. Это может быть текст в произвольном формате: на естественном языке, на языке программирования, на формальном логическом языке, например, OCL и т. д. Более того, если возможности инструмента это позволяют, в примечаниях можно хранить гиперссылки, вложенные файлы и другие артефакты, внешние по отношению к модели.

Примечания могут иметь стереотипы. В UML определены два стандартных стереотипа для примечаний:

- «requirement» — описывает общее требование к системе;
- «responsibility» — описывает ответственность сущности (классификатора).

Примечания первого типа часто присутствуют на диаграммах использования, а примечания второго типа — на диаграммах классов.

Возвращаясь к нашему примеру, будет совсем не лишним указать, что информацию о состоянии кадров нужно хранить постоянно, т. е. она не должна исчезать после завершения сеанса работы с системой (рис. 2.3).

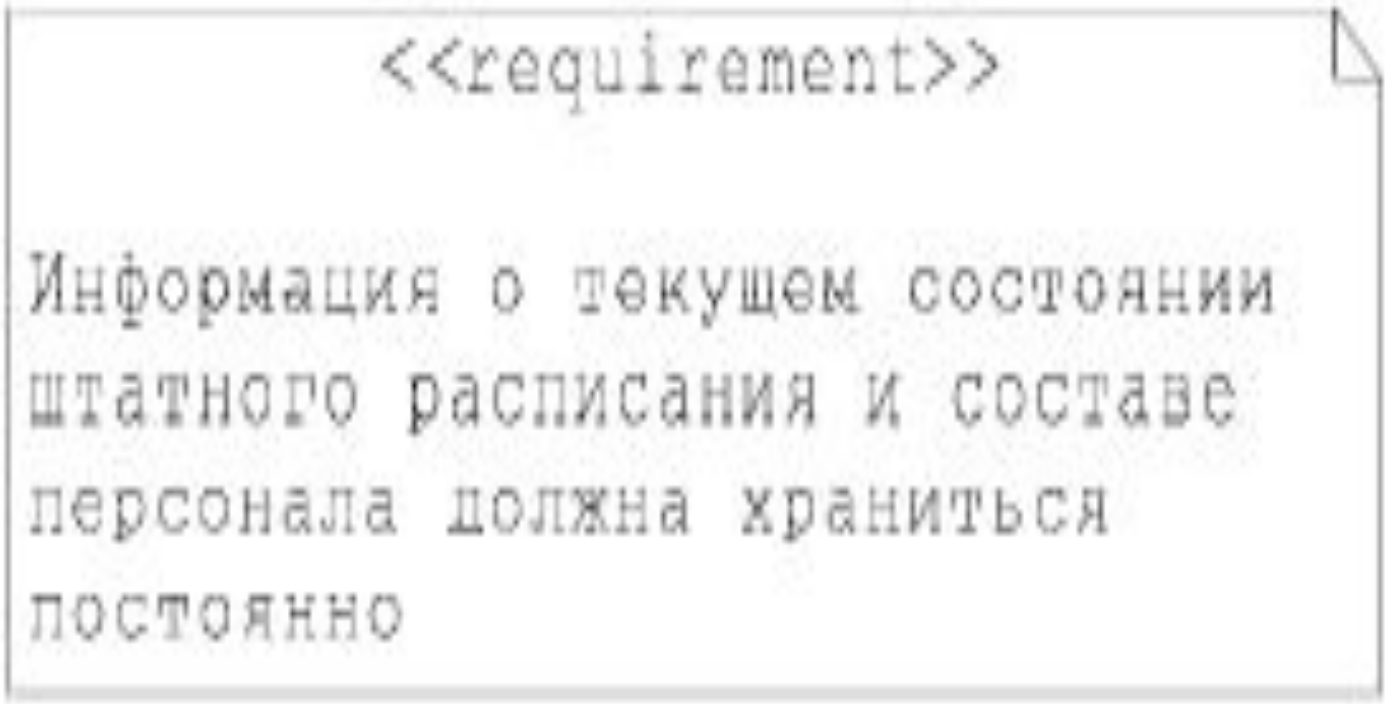


Рис. 2.3. Нефункциональное требование к ИС ОК

2.2.4. Отношения на диаграммах использования

Как уже было отмечено в первой главе, на диаграммах использования применяются следующие основные типы отношений:

- ассоциация между действующим лицом и вариантом использования;
- обобщение между действующими лицами;
- обобщение между вариантами использования;
- зависимости между вариантами использования.

Ассоциация между действующим лицом и вариантом использования *показывает, что действующее лицо тем или иным способом взаимодействует (предоставляет исходные данные, получает результат) с вариантом использования.*

Другими словами, эта ассоциация обозначает, что действующее лицо так или иначе, но обязательно непосредственно участвует в выполнении каждого из сценариев, описываемых вариантом использования. Ассоциация является наиболее важным и, фактически, обязательным отношением на диаграмме использования. Действительно, если на диаграмме использования нет ассоциаций между действующими лицами и вариантами использования, то это означает, что система не взаимодействует с внешним миром. Такие системы, равно как и их модели, не имеют практического смысла.

Применительно к нашему примеру в первом приближении можно обозначить ассоциации, представленные на рис. 2.4.

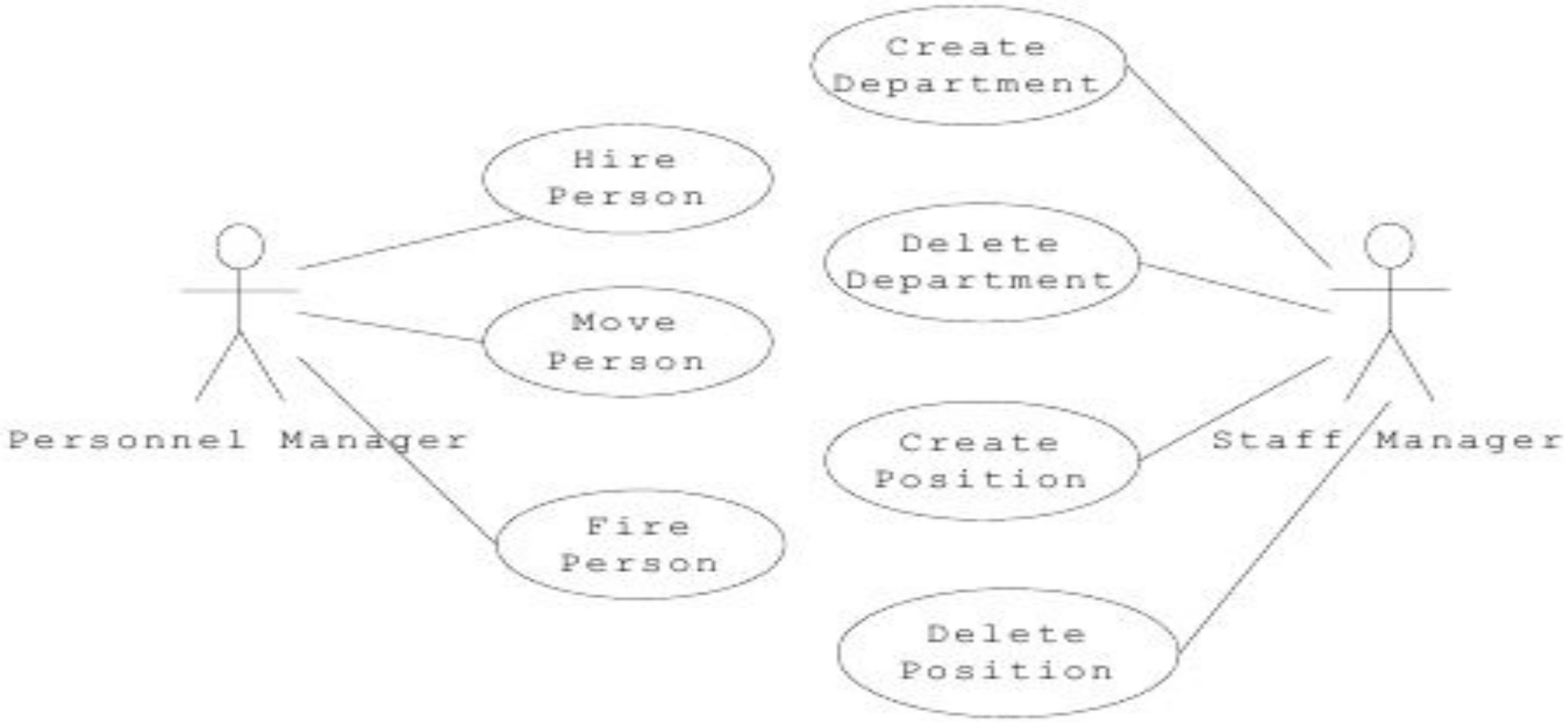


Рис. 2.4. Ассоциации между действующими лицами и вариантами использования

Обобщение между действующими лицами *показывает, что одно действующее лицо наследует все свойства (в частности, участие в ассоциациях) другого действующего лица.*

С помощью обобщения между действующими лицами легко показать иерархию категорий пользователей системы, в частности, иерархию прав доступа к выполняемым функциям и хранимым данным.

ИЗМЕНЕНИЯ В ТЕХНИЧЕСКОМ ЗАДАНИИ

Среди всех пользователей информационной системы следует выделить особую категорию пользователей (высшее руководство), которой разрешен доступ к любым данным и операциям.

Это изменение в требованиях можно отразить в модели системы так, как показано на рис. 2.5.

Рис. 2.5. Иерархия категорий пользователей ИС ОК

Действующее лицо, будучи классификатором, может быть абстрактным классификатором, то есть таким классификатором, который не может иметь непосредственных экземпляров. Введение абстрактных действующих лиц позволяет без потери информации сократить количество непосредственных ассоциаций в модели, сделав ее более лаконичной, а значит более наглядной и полезной.

ИЗМЕНЕНИЯ В ТЕХНИЧЕСКОМ ЗАДАНИИ

Информационная система должна предоставлять возможность просматривать данные без внесения в них каких-либо изменений.

Данное требование следует оформить в виде дополнительного варианта использования — `Browse`. Разумно предположить, что просматривать данные могут все категории пользователей. В этом случае можно поступить так, как показано на рис. 2.6, т. е. ввести обобщенного абстрактного пользователя `User` (1), который будет связан ассоциацией с вариантом использования `Browse` (2). При этом все специализации (3 и 4) обобщенного пользователя автоматически будут связаны ассоциацией с вариантом использования `Browse`.

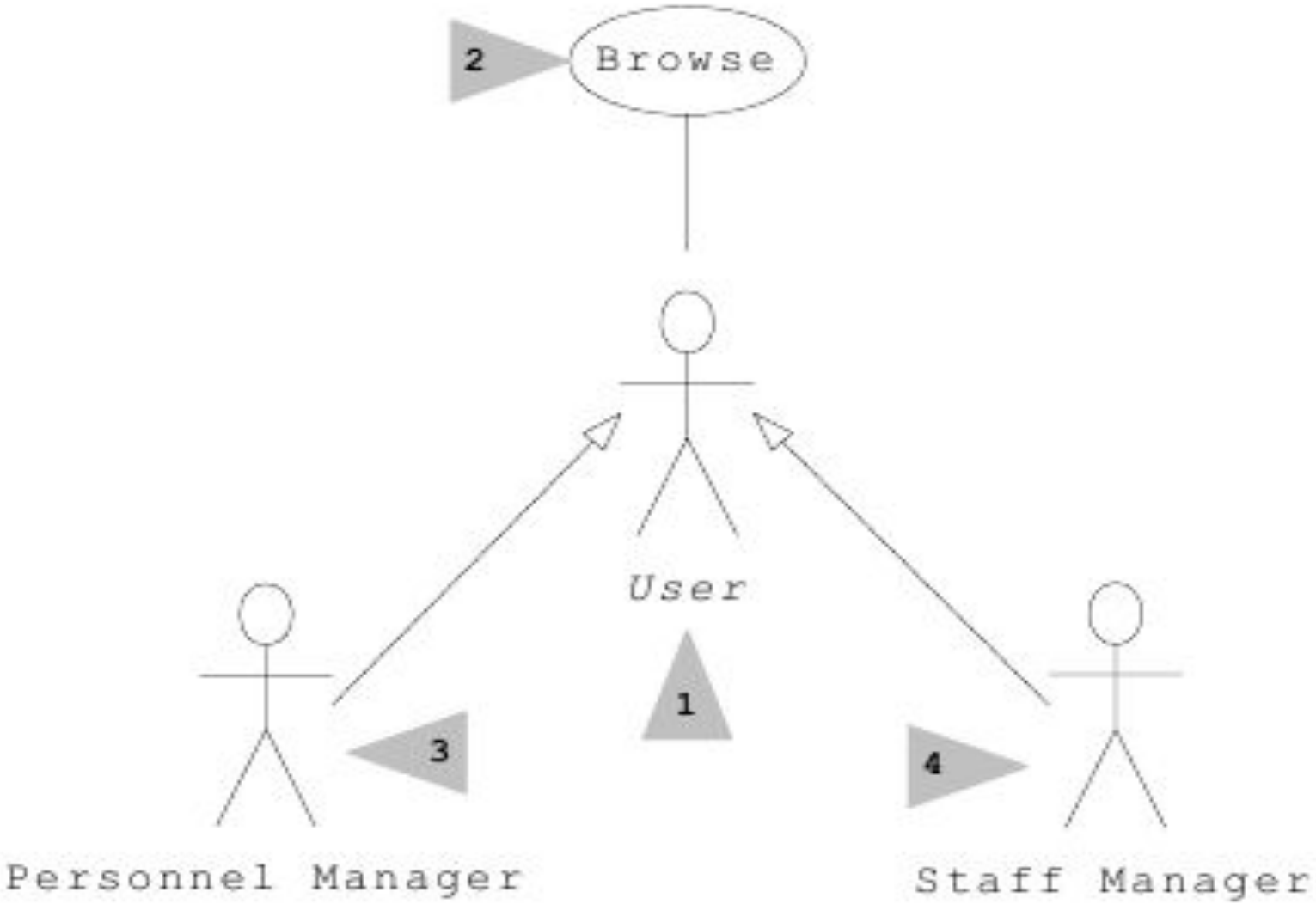


Рис. 2.6. Абстрактное действующее лицо

Обобщение между вариантами использования *показывает, что один вариант использования является частным случаем (подмножеством множества сценариев) другого варианта использования.*

Обобщающий вариант использования, будучи классификатором, может быть абстрактным классификатором. Например, такой важный для сотрудника вариант использования, как увольнение, на самом деле является абстракцией: в каждом конкретном случае имеет место ровно один из возможных частных случаев увольнения, которые приводят к одному и тому же результату с точки зрения менеджера персонала, но весьма различны с точки зрения сотрудника.

ИЗМЕНЕНИЯ В ТЕХНИЧЕСКОМ ЗАДАНИИ

Система должна поддерживать два способа увольнения сотрудника: по инициативе администрации и по собственному желанию.

Данное обстоятельство можно отразить в модели так, как показано на рис. 2.7.

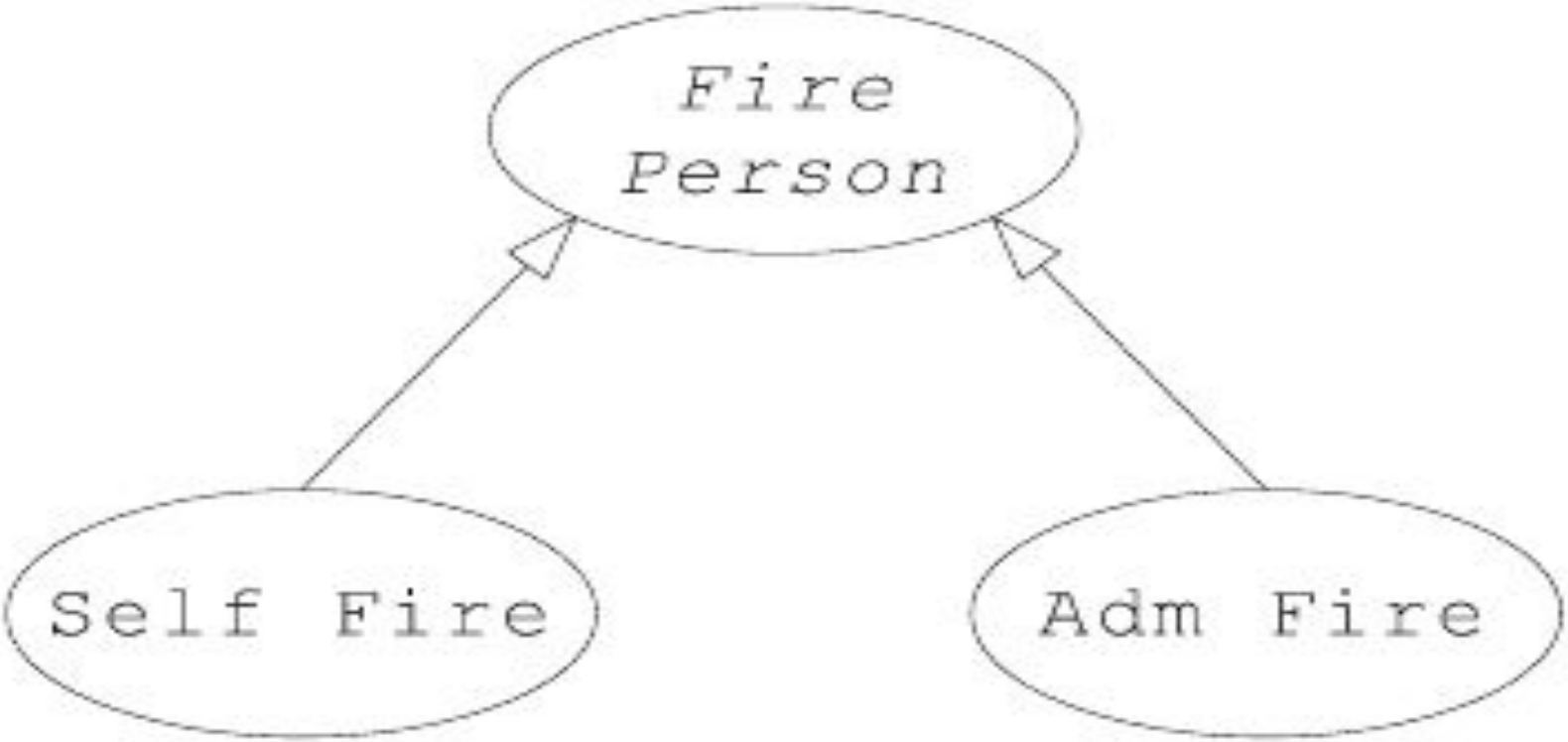


Рис. 2.7. Обобщение вариантов использования

Обобщенный абстрактный (имя написано курсивом) вариант использования `Fire Person` имеет две специализации, которые соответствуют увольнению работника по собственному желанию (`Self Fire`) и по инициативе администрации (`Adm Fire`).

Зависимость между вариантами использования *показывает, что один вариант использования зависит от другого варианта использования.*

В UML имеются два стандартных стереотипа зависимости между вариантами использования, которые в некотором смысле двойственны друг другу:

- «include» — показывает, что в **каждый** сценарий зависимого варианта использования в определенном месте вставляется в качестве подпоследовательности действий в сценарий независимого варианта использования;

- «extend» — показывает, что в **некоторый** сценарий независимого варианта использования может быть в определенном месте вставлен в качестве подпоследовательности действий сценарий зависимого варианта использования.

ИЗМЕНЕНИЯ В ТЕХНИЧЕСКОМ ЗАДАНИИ

При увольнении сотрудника должна быть осуществлена выплата денежной компенсации за неиспользованный отпуск. В случае вынужденного сокращения возможна выплата выходного пособия. Учетная запись сотрудника при увольнении должна быть заблокирована.

Блокировка учетной записи и выплата компенсации — это примеры вариантов использования, которые вполне могут быть востребованы как при увольнении, так и помимо него. Отношения зависимости между этими вариантами использования могут быть показаны на диаграмме использования, например, так, как это сделано на рис. 2.8.

Рис. 2.8. Зависимости между вариантами использования

2.2.5. Способы применения моделей использования

Если посмотреть на модель использования с самой общей точки зрения, то нетрудно заметить, что в модели присутствуют:

- внутренняя моделируемая система, в форме набора вариантов использования, возможно связанных зависимостями и обобщениями;
- внешнее окружение, в форме набора действующих лиц, возможно связанных обобщениями;

- связь между моделируемой системой и внешним окружением в форме ассоциаций между действующими лицами и вариантами использования.

Обычно совершенно ясно, что находится внутри моделируемой системы, а что снаружи. Если это почему-либо неясно, или же требуется увеличить наглядность диаграмм, то можно воспользоваться специальной конструкцией, которая называется "границы системы".

Границы системы *(system boundary) — это графический комментарий в форме прямоугольной рамки, применяемый на диаграммах использования и отделяющий внутреннюю часть системы от ее внешнего окружения.*

Внутренняя часть, выделяемая границами, имеет в UML конкретное название — субъект.

Субъект *(subject) — это классификатор, который реализует поведение, декларируемое вариантами использования.*

Если границы системы используются на диаграмме, то можно указать имя (и стереотип!), которые будут относиться к субъекту[5]. В примере на рис. 2.9 мы повторили рис. 2.4, но использовали другие возможности нотации UML.

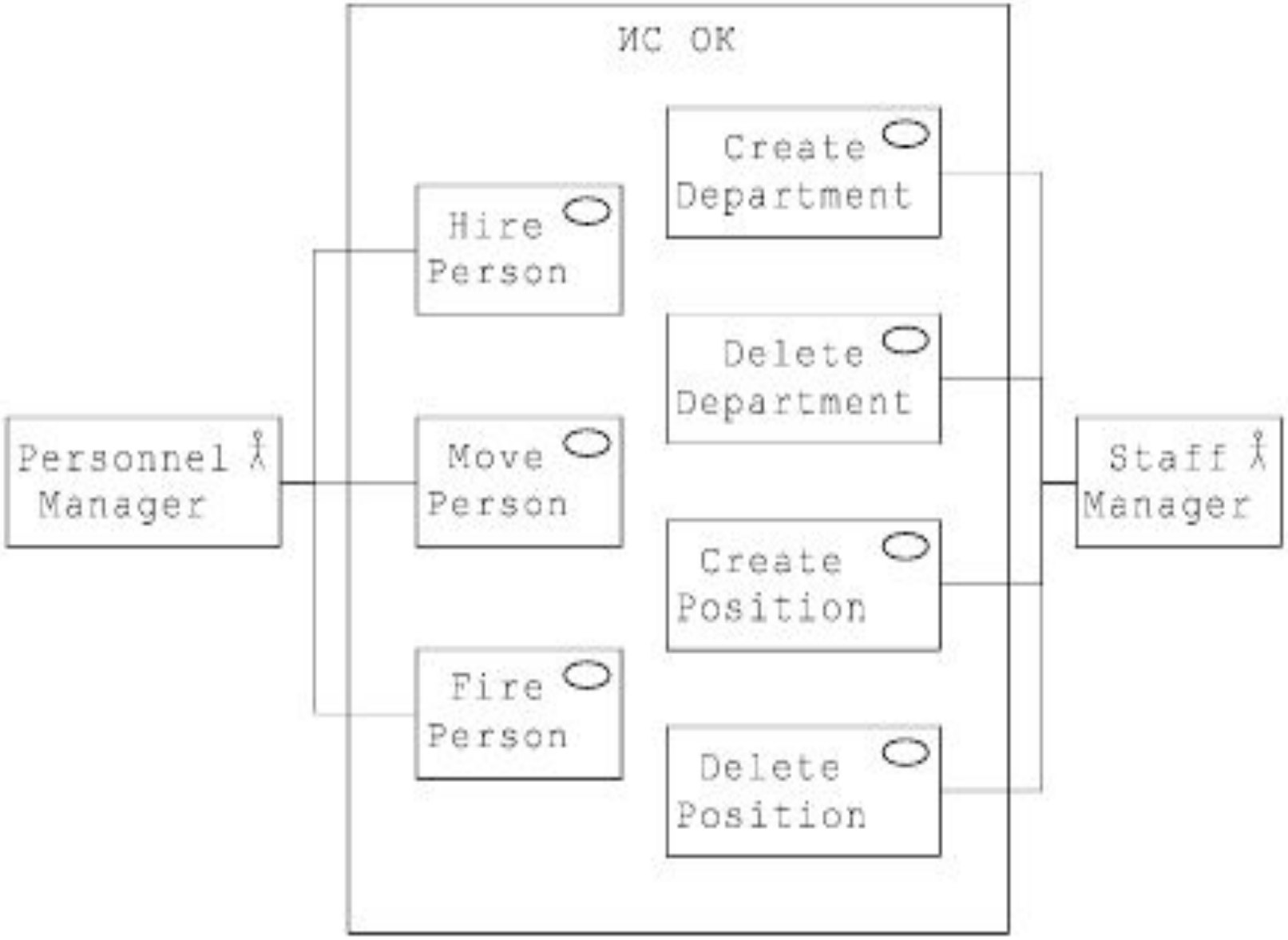


Рис. 2.9. Границы системы

2.3. РЕАЛИЗАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

После того, как построено представление использования (результат моделирования использования), то есть, выделены действующие лица, варианты использования и установлены отношения между ними, встает естественный вопрос: что дальше? То есть, как далее следует продолжать моделирование средствами UML?

Действующие лица находятся вне системы — с ними ничего делать не нужно. Можно сказать, что действующие лица уже выполнили свою задачу, просто появившись в модели системы. Таким образом, переход от моделирования использования к другим видам моделирования состоит в уточнении, детализации и конкретизации вариантов использования. В представлении использования мы показали, **что** делает система, теперь нужно определить, **как** это делается. Это обычно называется реализацией вариантов использования.

Реализация варианта использования *(use case realization) — это описание всех или некоторых сценариев, составляющих вариант использования.*

Повторим еще раз: вариант использования — это описание множества последовательностей событий или действий (сценариев), доставляющих значимый для действующего лица результат. Наиболее часто используемый метод описания множества последовательностей действий состоит в указании *алгоритма*, выполнение которого доставляет последовательность действий из требуемого множества[6]. Существует множество способов описания алгоритмов, более или менее формальных. Мы рассмотрим четыре, часто применяемых именно при реализации вариантов использования.

2.3.1. Текстовые описания

Исторически самый заслуженный и до сих пор один из самых популярных способов: составить текстовое описание типичного сценария варианта использования.

Рассмотрим следующий ниже текст в качестве примера одного из возможных сценариев.

Сценарий варианта использования *Увольнение по собственному желанию*

Сценарий варианта использования *Увольнение по собственному желанию*

1. Сотрудник пишет заявление

2. Начальник подписывает заявление

3. ***Если*** *есть неиспользованный отпуск,*

то *бухгалтерия рассчитывает компенсацию*

4. Бухгалтерия рассчитывает выходное пособие

5. Системный администратор удаляет учетную запись

6. Менеджер персонала обновляет базу данных

---

[6] Последовательность действий при конкретном выполнении алгоритма называется *протоколом* этого *выполнения*. Таким образом, сценарий можно рассматривать как протокол выполнения алгоритма варианта использования.

Казалось бы, что здесь неясного? А неясно, например, вот что: как должна вести себя система, если на шаге 2 начальник *не* подписывает заявление. Из текста сценария не только не ясен ответ, но, хуже того, при невнимательном чтении можно и не заметить, что есть вопрос.

Текстовые описания сценариев всем хороши: просты, всем понятны, легко и быстро составляются. Плохи они тем, что могут быть неполны и неточны, и эти недостатки незаметны.

2.3.2. Реализация программой на псевдокоде

Второй рассматриваемый нами способ реализации варианта использования — записать алгоритм на псевдокоде. Этот способ хорош тем, что понятен, привычен и доступен любому разработчику. Однако в настоящее время вряд ли можно рекомендовать такой способ реализации, как основной, по следующим причинам.

1. Реализация на псевдокоде плохо согласуется с современной парадигмой объектно-ориентированного программирования.

2. При использовании псевдокода теряются все преимущества использования UML: наглядная визуализация с помощью картинки, строгость и точность языка проектирования и реализации, поддержка распространенными инструментальными средствами.

3. Решения на псевдокоде практически невозможно использовать повторно.

Тем не менее, рассмотрим пример реализации вариантов использования на псевдокоде.

```
Use case Self Fire

Получить заявление

add_payment:

Pay Compensation(Self Fire, add_payment)

Include Delete Account

Обновить информацию в базе данных
```

Use case Adm Fire

Получить приказ

add_payment:

Pay Compensation(Adm Fire, add_payment)

Include Delete Account

Обновить информацию в базе данных

Use case Pay Compensation

```
if (add_payment)
        if (from Self Fire)
            начислить за неиспользованный отпуск
        else if (from Adm Fire)
            начислить выходное пособие
```

Увольнение по собственному желанию запускается по инициативе сотрудника. Увольнение по инициативе администрации начинается с приказа об увольнении. В остальном последовательность действий в обоих случаях совпадает.

В этих текстах использовано ключевое слово `Include`, отражающее наличие зависимостей с таким стереотипом в модели. А именно, это означает, что в этом месте в текст псевдокода для данного варианта использования нужно включить текст псевдокода для варианта использования `Delete Account`, который мы здесь не приводим.

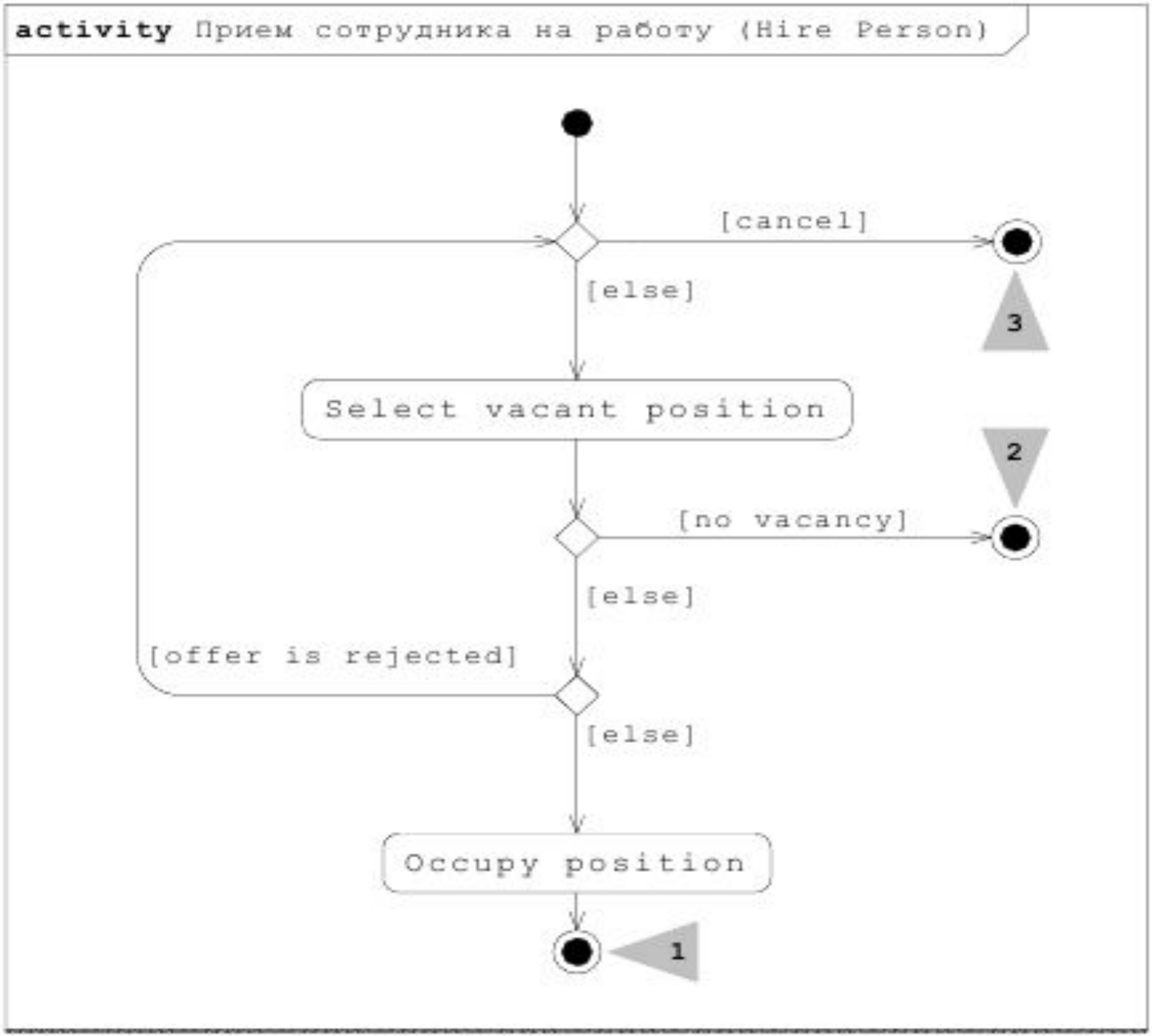
Вариант использования Pay Compensation запускается, если есть условия для выплаты компенсаций. При этом основные варианты использования не должны знать, каковы эти условия и как рассчитывается компенсация — за это отвечает вариант использования Pay Compensation Зависимость со стереотипом «extend» означает, что псевдокод варианта использования Pay Compensation должен быть включен в текст основных вариантов использования. При этом вариант использования Pay compensation должен знать, в какое место ему нужно включиться. Для этого в основных вариантах использования определена *точка расширения* (extension point) — по сути, просто метка в программе.[7] Этот пример в достаточной мере объясняет сходство и различие между зависимостями со стереотипом «include» и «extend».

2.3.3. Реализация диаграммами деятельности

Третий способ реализации варианта использования — описать алгоритм с помощью диаграммы деятельности. С одной стороны, диаграмма деятельности — это полноценная диаграмма UML, с другой стороны, диаграмма деятельности немногим отличается от блок-схемы (а тем самым и от псевдокода). Таким образом, реализация варианта использования диаграммой деятельности является компромиссным способом ведения разработки — в сущности, это проектирование сверху вниз в терминах и обозначениях UML.

Например, в информационной системе отдела кадров прием сотрудника может быть организован так, как показано на рис. 2.10.

Приблизило ли нас появление этой диаграммы в модели к завершению работы над системой? С одной стороны, вместо одной сущности, подлежащей реализации (вариант использования Hire Person) появилось пять новых: три сторожевых условия и две деятельности, которые в свою очередь теперь нуждаются в реализации. С другой стороны, каждая из этих новых сущностей кажется более простой и понятной, а значит быстрее и надежнее реализуемой. Кроме того, эту диаграмму можно показать заказчику, чтобы проверить, действительно ли проектируемая нами логика работы системы соответствует тому бизнес-процессу, который существует в реальности.


activity Прием сотрудника на работу (Hire Person)
[cancel]
[else]
3
Select vacant position
2
[no vacancy]
[else]
[offer is rejected]
[else]
Occupy position
1

Применение диаграмм деятельности для реализации вариантов использования может привести к более глубокому пониманию существа задачи и даже открыть неожиданные возможности улучшения приложения, которые было трудно усмотреть в первоначальной постановке задачи.

Например, рассматривая (чисто формально) схему процесса на рис. 2.10, мы видим, что процесс может иметь три исхода.

1. Нормальное завершение (1 на рис. 2.1), которое предполагает обязательное выполнение деятельности `Occupy position`. Резонно предположить, что при выполнении этой деятельности в базу данных будет записана какая-то информация, что, несомненно, является значимым для пользователя результатом.

2. Исключительная ситуация (2 на рис. 2.10), возникающая в том случае, когда процесс не может быть нормально завершен, т. е. мы хотели принять человека на работу, и он был согласен, а текущее состояние штатного расписание не позволило этого сделать. Факт возникновения такой ситуации, хотя формально и не является значимым результатом для менеджера персонала, на самом деле может быть весьма важен для высшего руководства или менеджера штатного расписания.

3. Завершение процесса без достижения какого-либо видимого результата (3 на рис. 2.10). Например, менеджер персонала сделал кандидату какие-то предложения, но ни одно из них кандидата не устроило, после чего они расстались, как будто ничего и не было.

Этот простой анализ наталкивает на следующие соображения. Вариант использования **должен** доставлять значимый результат, значит, если результата нет, то что-то спроектировано не так, как нужно. Действительно, все практические информационные системы отдела кадров обязательно накапливают статистическую информацию обо всех **проведенных** кадровых операциях. Такая статистика

обо всех **проведенных** кадровых операциях. Такая статистика необходима для анализа движения кадров. Однако далеко не все системы позволяют учитывать и **не проведенные** операции. Между тем, учет причин, по которым кандидаты не принимают предложенной работы или, наоборот, причин, по которым организации отвергают кандидатов, может быть весьма полезен. Поэтому мы можем усовершенствовать нашу диаграмму, например, так, как показано на рис. 2.11.

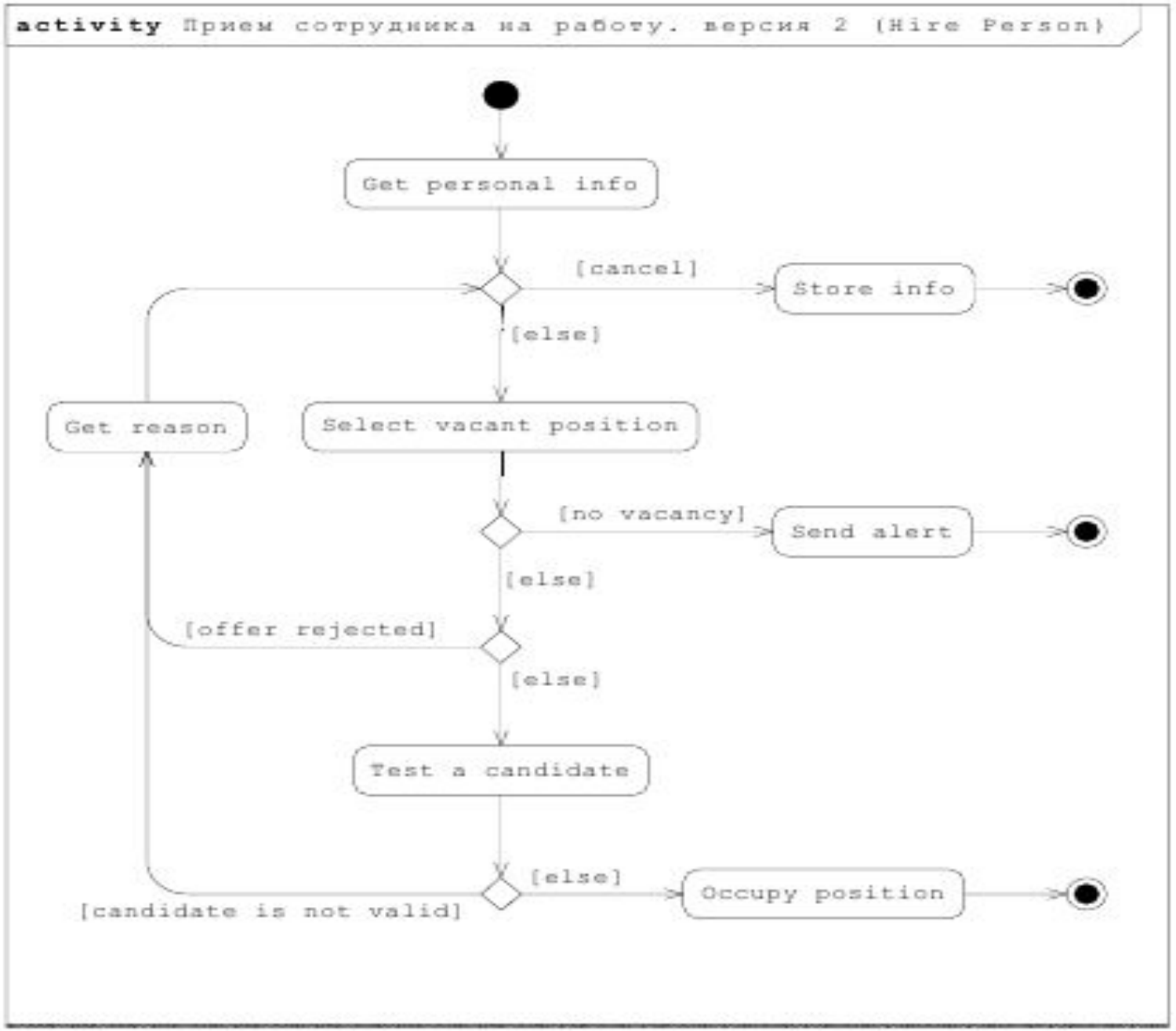


Рис. 2.11. Усовершенствованная реализация варианта использования

Вот теперь (формально) все хорошо: информация не теряется. Более того, имеет смысл вернуться к представлению использования и посмотреть, не нужно ли включить в модель новые варианты использования (может быть, как низкоприоритетные и подлежащие реализации в последующих версиях системы). Так реализация вариантов использования может приводить к изменению и усовершенствованию самих вариантов использования. **Моделирование имеет итеративный характер**, о чем мы уже говорили в главе 1.

2.3.4. Реализация диаграммами взаимодействия

Четвертый из основных способов реализации варианта использования — создать одну или несколько диаграмм взаимодействия в форме диаграмм коммуникации или диаграмм последовательности, которые описывают один или несколько сценариев данного варианта использования. Этот способ в наибольшей степени соответствует идеологии UML и рекомендуется авторами языка как основной и предпочтительный.

Рассмотрим пример реализации диаграммами взаимодействия варианта использования `Hire Person` (прием сотрудника на работу) информационной системы отдела кадров.

Сначала рассмотрим типовой сценарий, когда прием проходит безо всяких осложнений: есть вакантное рабочее место и кандидат готов его занять. Диаграмма последовательности для такого сценария приведена на рис. 2.12.

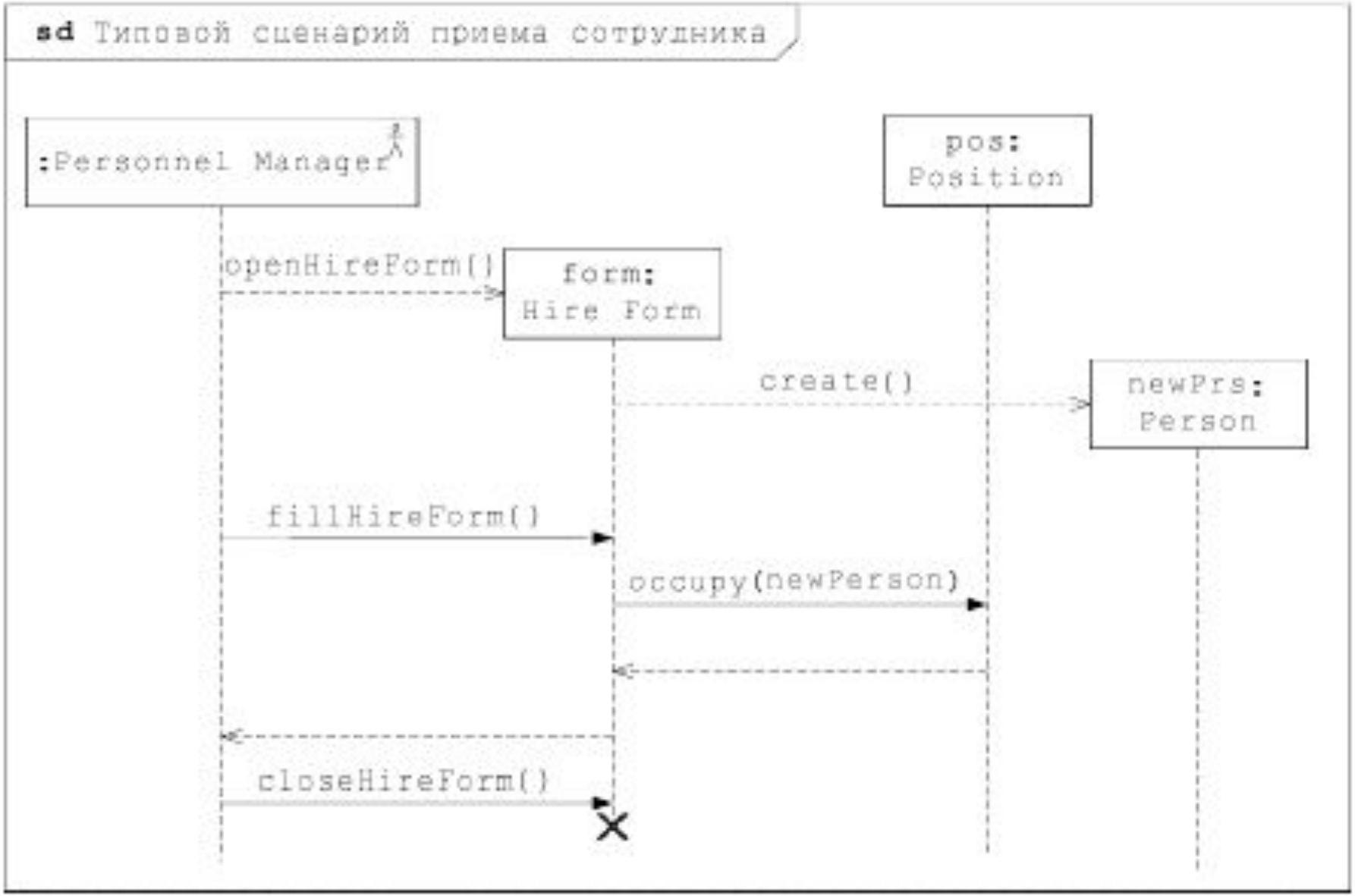


Рис. 2.12. Диаграмма последовательности для типового сценария

На приведенной диаграмме последовательность посылаемых сообщений примерно соответствует последовательности действий на диаграмме деятельности (см. рис. 2.11) в том случае, когда поток управления проходит по диаграмме сверху вниз один раз. Таким образом, диаграмма, представленная на рис. 2.12 до некоторой степени определяет типовой сценарий варианта использования `Hire Person`. Однако построив такую диаграмму, мы постулировали существование в системе некоторых классов, экземпляры которых должны взаимодействовать для обеспечения требуемого поведения моделируемого варианта использования. Здесь в нашей модели появились новые сущности:

появились новые сущности:

- класс `Hire Form`, ответственный за интерфейс, необходимый для выполнения варианта использования прием сотрудника;
- класс `Person`, ответственный за хранение данных о конкретном сотруднике;
- класс `Position`, ответственный за хранение данных и выполнение операций с конкретной должностью.

Возвращаясь к нашему примеру, заметим, что диаграмма на рис. 2.12 семантически не полна: она не отражает все сценарии варианта использования, которые предусматриваются, например, на диаграмме на рис. 2.11. Как уже было сказано, в этом случае можно составить дополнительные диаграммы взаимодействия, реализующие альтернативные сценарии варианта использования. Например, на рис. 2.13 показан сценарий приема сотрудника, соответствующий исключительной ситуации, когда нет вакантных должностей. На этот раз мы описываем сценарий в форме диаграммы коммуникации.

Из материала этого раздела мы делаем следующий вывод: **реализация вариантов использования диаграммами взаимодействия является наиболее трудоемким и сложным методом, но этот метод лучше всего согласован с объектно-ориентированным подходом и в наибольшей мере приближает нас к конечной цели.**

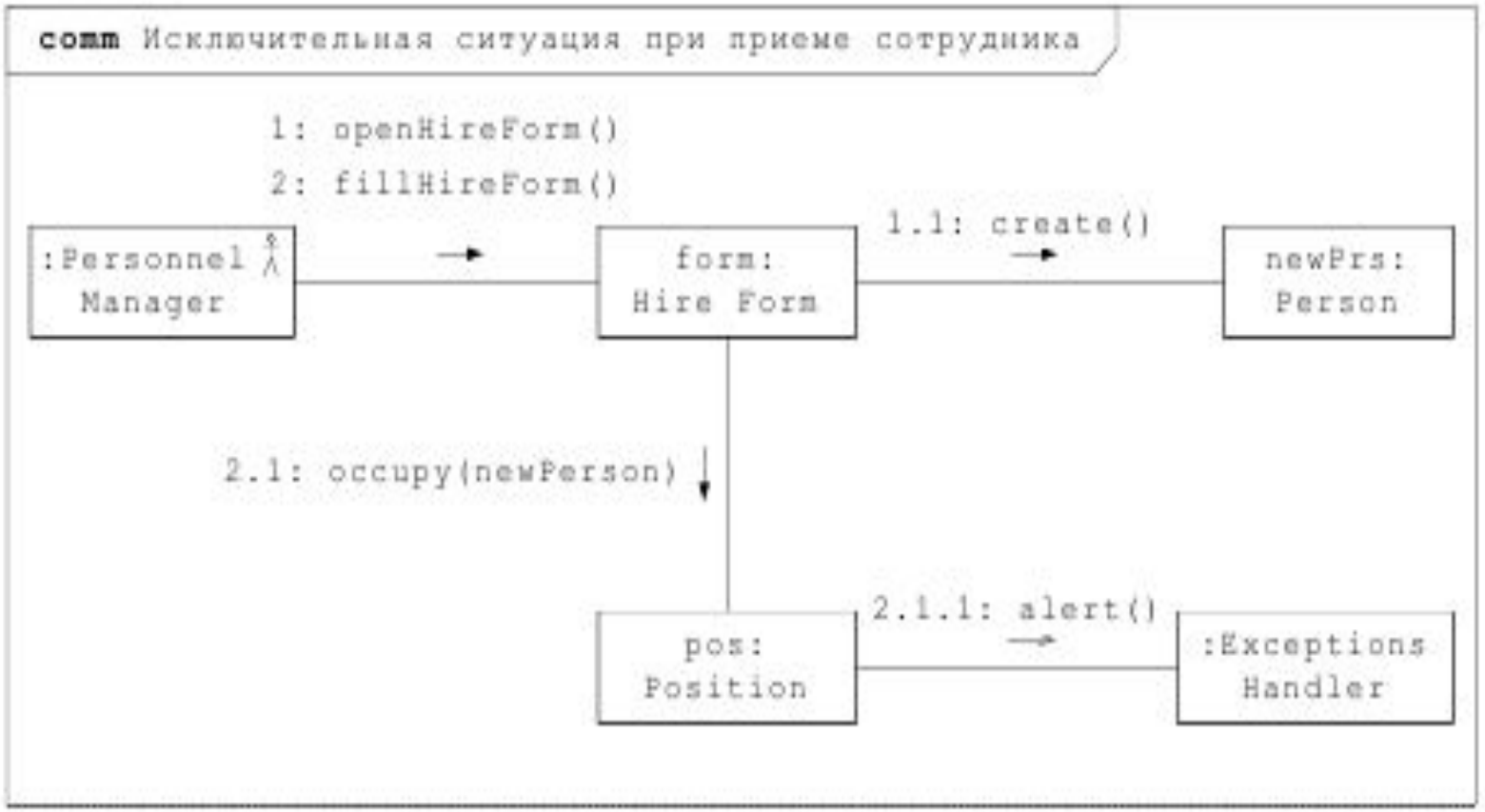


Рис. 2.13. Диаграмма кооперации для исключительной ситуации

ВЫВОДЫ

1. Составление диаграмм использования — это первый шаг моделирования.

2. Основное назначение диаграммы использования — показать, что делает система во внешнем мире.

3. Диаграмма использования не зависит от программной реализации системы и поэтому не обязана соответствовать структуре классов, модулей и компонентов системы.

4. Идентификация действующих лиц и вариантов использования — ключ к дальнейшему проектированию.

5. В зависимости от выбранной парадигмы проектирования и программирования применяются различные способы реализации вариантов использования.