Использование ролей в сценариях взаимодействия

А. Волков (volkov@ispras.ru)

Аннотация. Сценарий взаимодействия описывает поведение системы через описания взаимодействий некоторых объектов. Такими объектами выступают компоненты системы и внешние объекты, называемые *акторами* (*actors*). Обычно в сценарии взаимодействия подразумевается уникальность каждого объекта системы, т.е. предполагается, что описываются физически различные экземпляры объектов. Однако в ряде случаев это приводит к дублированию при описании сценариев. В работе предлагается расширить аппарат сценариев взаимодействия, разрешив использование ролей объектов. Роль соответствует некоторому срезу поведения определенного физического объекта. При таком подходе к описанию поведения возникает задача композиции поведения объектов в различных ролях. В данной статье рассматривается возможность систематического использования понятия роли в сценариях и исследуются средства, позволяющие строить общее поведение для объектов моделируемой системы по их поведению в различных ролях. Для представления сценариев используется модель взаимодействий UML (UML Interactions). В качестве абстрактных моделей для описания общего поведения объектов рассматриваются автоматные модели и модели, основанные на сетях Петри, в нотации UML (машины состояний и активности соответственно).

1. Введение

Модельно-ориентированный подход (Model Driven Architecture, MDA), предложенный консорциумом OMG, направлен на достижение интероперабельности разрабатываемых систем. В нем используется идея разделения бизнес-логики проектируемой системы и конкретной технологии ее реализации. Для описания бизнес-логики используются платформо-независимые модели проектируемой системы. В рамках этого подхода основополагающую роль играет построение моделей системы, проверка их корректности и отображение в модели других уровней. Для унификации моделирования различных аспектов системы в рамках подхода MDA предлагается использовать универсальную нотацию – язык UML [1].

На ранних фазах проектирования программных системы, особенно на фазе анализа требований, с успехом используется сценарный подход, заключающийся в определении *вариантов использования* (*use cases*) системы и описания сценариев ее поведения в каждом таком варианте. Каждый сценарий представляет собой описание последовательности взаимодействий,

направленной на достижение некоторой цели. Сценарии могут быть заданы с помощью какой-либо нотации, позволяющей описывать поведение, однако, как правило, для описания сценариев используются нотации, обладающие высокой степенью наглядности.

Построение формализованной сценарной модели позволяет производить как статический анализ требований, так и генерировать исполняемый прототип системы для динамического исследования системы; тем самым достигается возможность проверки требований.

Для построения поведенческих моделей в языке UML 2.0 существуют следующие средства:

- **взаимодействия** (*interactions*), семантика которых задает отношение частичной упорядоченности событий в различных компонентах системы и акторов,
- **активности** (*activities*), семантически эквивалентные иерархическим раскрашенным сетям Петри,
- **машины состояний** (*state machines*), использующие семантику расширенных конечных автоматов в алфавите событий системы.

По сути, взаимодействия UML обеспечивают абстракцию трасс системы, активности – абстракцию потоков (управления и данных) в системе, а машины состояний – абстракцию последовательности состояний для каждого компонента системы. Как правило, для описания сценариев на ранних фазах наиболее адекватно соответствует семантика взаимодействий UML, так как в сценариях, по существу, описываются трассы взаимодействий с будущей системой. Далее, говоря о сценариях, мы будем подразумевать, что они описываются в рамках модели взаимодействий UML.

Семантика моделей взаимодействий носит декларативный характер. При определенных ограничениях возможно статическое исследование таких моделей (например, см. в [2] использование линеаризаций последовательностей событий и анализ формальных языков, получающихся в результате их рассмотрения). Для определения исполняемой семантики модели взаимодействий возможен подход, заключающийся в построении специальной абстрактной машины (вариант такого построения можно найти в [3]). Однако при составлении требований основной акцент делается, как правило, на описание ожидаемых ситуаций в системе. Кроме того, поведение описывается для некоторой совокупности объектов системы совместно, в то время как во многих случаях подразумевается задание протокола взаимодействия существенно независимых объектов. Абстрактная реализация модели поведения будет полностью следовать описанным сценариям, однако для указанных систем в ходе разработки сценариев может остаться неучтенной это существенная независимость объектов системы.

В отличие от моделирования требований, для описания прототипа системы строятся, как правило, модели, описывающие поведение каждого объекта системы в отдельности; такие модели могут быть с легкостью отображены в

целевой код прототипа. Для описания этих моделей хорошо подходят событийные автоматы, которым соответствуют машины состояний UML. Различие в поведении такого прототипа системы и поведении, задаваемом моделью требований в виде взаимодействий, в некоторых случаях свидетельствует о недопонимании требований к системе.

Таким образом, для анализа требований системы в форме сценариев взаимодействия оказывается полезным исследовать поведение системы, которое описывается моделями, отличными от модели последовательностей. Для этого важна возможность преобразования одних модели поведения в другие. При этом встает вопрос о соотношениях между различными моделями поведения системы, т.е. вопрос об эквивалентности поведений, задаваемых разными моделями или, по крайней мере, о степени точности аппроксимации поведения одной модели поведением другой. Исследование неточностей, появляющихся в результате такой аппроксимации может помочь выявить те особенности подразумеваемого поведения системы, которые не были учтены в исходных сценариях.

Привлекательной является возможность автоматического преобразования исходной сценарной спецификации, моделируемой последовательностями UML, в другие модели, аппроксимирующие исходную.

На текущий момент существует алгоритм построения по сценариям, записанным в нотации диаграмм последовательностей, автоматной модели для каждого объекта, входящего в систему; совокупность этих автоматов аппроксимирует исходную модель. Этот алгоритм реализован в семействе инструментов Bridge, разрабатываемых в Институте Системного программирования РАН совместно с компанией Klocwork Inc. Данная работа ведется в рамках этого проекта.

Для описания UML-взаимодействий в данной работе используются *диаграммы последовательностей* (*sequence diagrams*) UML, а для описания композиции поведения – *обзорные диаграммы взаимодействий* (*interaction overview diagrams*). Заметим, что сходную нотацию предоставляет стандарт MSC [4], который включает *базовые (basic*) и *высокоуровневые* (*high-level*) *MSC-диаграммы*. Отметим, что обзорные диаграммы взаимодействий имеют в некоторых случаях недостаточно определенную семантику для адекватного объединения поведения компонентов системы в различных сценариях, что затрудняет и статический анализ требований.

Каждое описание взаимодействий производится для определенного контекста, который определяется набором экземпляров объектов, принадлежащих системе, и внешними акторами. Возможность абстракции такого контекста от остальной системы может быть ценной при необходимости повторного описания таких взаимодействий для другого аналогичного контекста, так как тогда расширяется возможность *повторного использования* (*reusability*) описанных взаимодействий. Если в качестве абстракции экземпляров объектов традиционно рассматриваются классы объектов, то в качестве абстракции

объектов в рамках определенного контекста могут быть рассмотрены роли. В спецификации UML явно говорится о том, что проявления типов объектов в некотором контексте имеет смысл рассматривать как роли этих объектов.

Абстракция роли достаточно исследована со стороны ее структурной реализации, но не с точки зрения объединения поведения в различных сценариях.

Данная статья посвящена построению аппарата, позволяющего систематически использовать понятие роли при написании сценариев, определению средств нотации диаграмм последовательностей UML, которые могут быть использованы для этого, и исследованию того, каким образом полученная сценарная модель может быть отображена в другие модели.

Далее работа построена следующим образом:

- во втором разделе приводятся примеры, иллюстрирующие использование понятия роли в моделях взаимодействия;
- в третьем разделе производится некоторая формализация понятия роли;
- в четвертом разделе рассматриваются средства нотации диаграмм последовательностей и обзорных диаграмм взаимодействия UML для описания композиции поведения объектов в различных ролях;
- пятый раздел посвящен исследованию возможности отображения модели взаимодействий в другие поведенческие модели, рассматривается расширение алгоритма синтеза событийных автоматов при использовании средств композиции поведения в различных ролях, рассмотренных перед этим.

2. Роли в моделях взаимодействия

Описание взаимодействий в виде диаграммы последовательностей представляет собой набор *осей* (*lifelines*), некоторым образом сопоставляемых объектам, каждая из которых задает порядок наступления определенных событий для соответствующего объекта и выполнения им некоторых действий.

Мы будем придерживаться следующего соглашения об использовании синтаксиса именования оси:

`<идентификатор_оси> ::= [<имя_элемента>] [: <имя_класса>]`

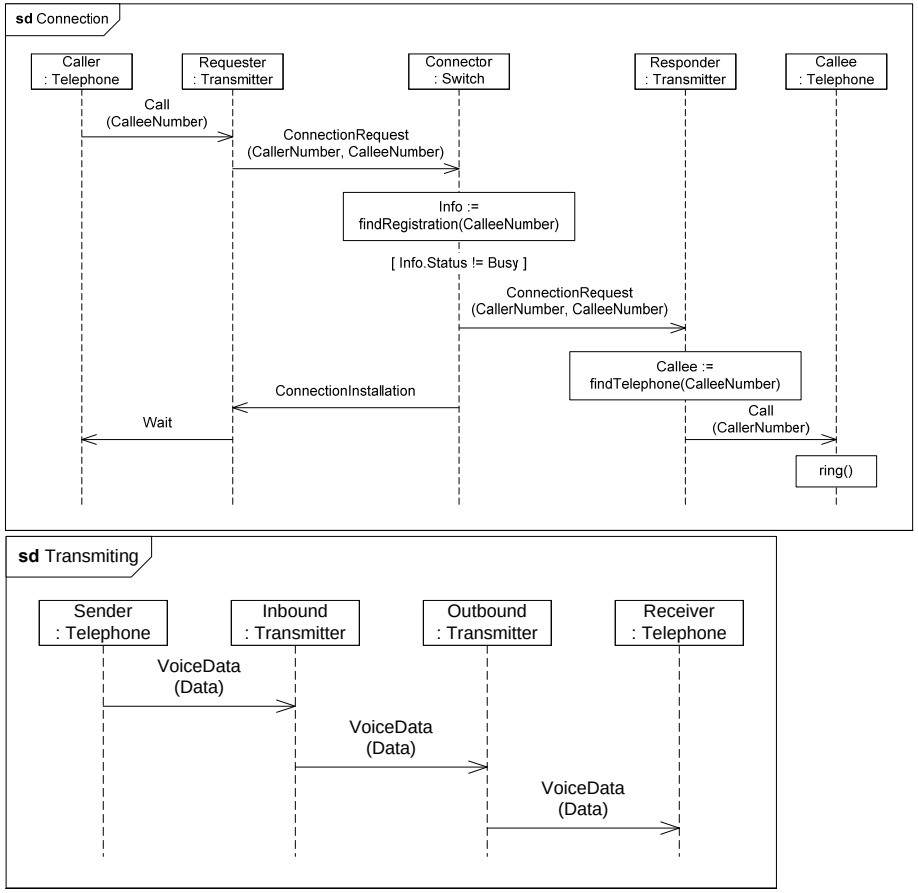
(причем `<идентификатор оси>` не может быть пустым).

Трактоваться `<идентификатор_оси>` будет следующим образом: `<имя_элемента>` обозначает имя роли объекта, которому данная ось ставится в соответствие, а `<имя_класса>` – тип объекта. В случае, когда опущено имя роли, предполагается, что объект играет некую, так называемую, *анонимную* роль. Если опущено имя класса, предполагается, что тип объекта может быть произвольным.

В качестве близкого к реальным системам примера рассмотрим модель сети мобильной связи, в которой есть телефоны (*Telephone*), передатчики (*Transmitter*) и центральный коммутатор (*Switch*). Для такой системы можно выделить следующие протоколы:

- установление соединения;
- передача данных (разговор);
- разрыв соединения;
- регистрация в сети;
- смена станции в сети.

Мы не будем описывать полную сценарную спецификацию для этой системы, однако приведем ее некоторые возможные сценарии (Рис. 1): запрос на установление соединения (*Connection*), передача данных (*Transmitting*), смена станции (*Relocation*).



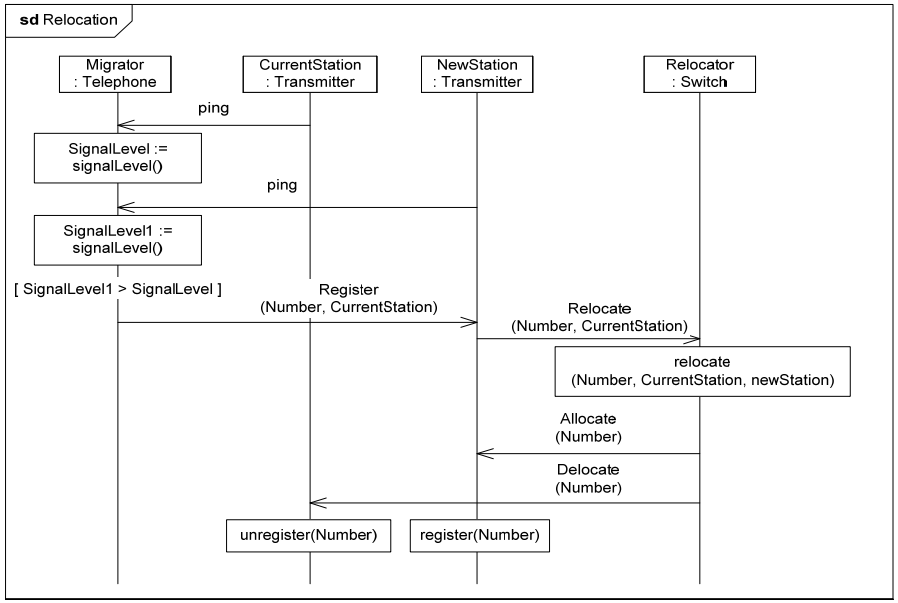


Рис. 1. Сценарии системы мобильной связи

Далее, во избежание громоздкости, в качестве модельного примера будет рассматриваться следующая спецификация достаточно простой системы, ее можно считать сильным упрощением описанной системы мобильной связи.

Итак, рассмотрим систему обмена данными (Рис. 2), состоящую из двух узлов `A` и `B`, которые могут по очереди обмениваться сообщениями с данными (*data*) до тех пор, пока один из них не пошлет сообщение о завершении серии обменов (*stop*).

По существу, в рассматриваемой спецификации имеются только два различных взаимодействия (Рис. 3; здесь опущены конкретные имена типов объектов) – это протокол передачи данных и протокол завершения обмена данными; для данного модельного примера эти протоколы тривиальны.

Протокол передачи данных описывает поведение двух сторон – объекта, играющего роль отправителя данных (*DataSender*), и объекта, играющего роль получателя данных (*DataReceiver*). В контексте завершения сеанса передачи можно говорить о роли объекта, инициирующего завершение (*TerminationInitiator*), и роли объекта, извещаемого о завершении (*TerminationAcceptor*). В этом примере следует обратить внимание, на то, что, прибегая к абстрагированию, можно говорить не об описании поведения объектов, а об описании поведения ролей, которые объекты могут выполнять.

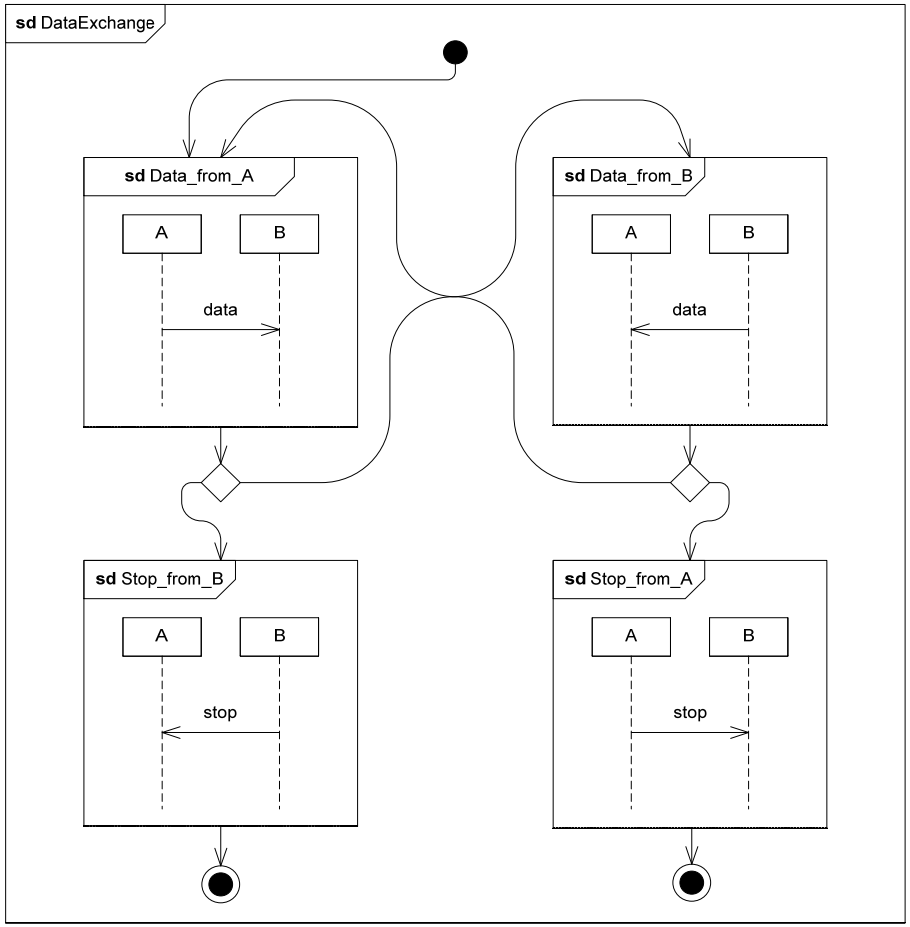


Рис. 2. Спецификация модельной системы обмена данными

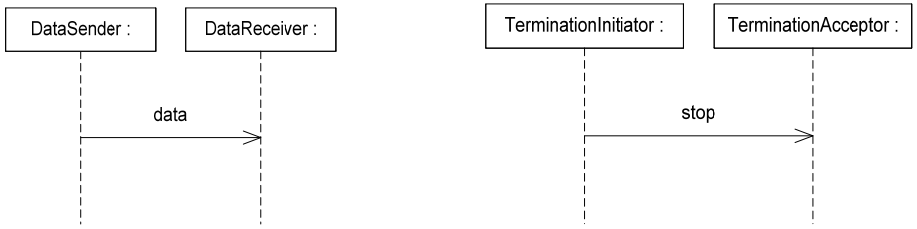


Рис. 3 Взаимодействия в системе обмена данными.

В данной спецификации эти протоколы пришлось продублировать для вариантов разных сочетаний объектов `A` и `B` и их возможных ролей.

Теперь рассмотрим следующие два вопроса:

- если объекты `A` и `B` – различных типов, то как избежать дублирования описания общего для них поведения?
- если объекты `A` и `B` одинакового типа, то как описать композицию поведения объектов этого типа?

С использованием понятия ролей эти вопросы могут быть переформулированы следующим образом:

- как для объектов различных типов, которые могут выполнять в некотором контексте одну и ту же роль, описывать включение поведения в этой роли в их суммарное поведение и строить такой результат?
- как для объектов одного типа, но выполняющих разные роли, описывать композицию поведений в них и строить результат этой композиции?

При успешном решении этих задач выделение понятия роли в сценариях позволит избежать дублирований при их описании. Случаи осмысленного выделения ролей, соответствуют ситуациям, когда в описаниях сценариев имеются:

- объекты одного типа, участвующие в одном и том же взаимодействии различным образом (т.е. в различных ролях);
- объекты разных типов, которые в некоторых взаимодействиях могут вести себя одинаково (т.е. выполнять одну и ту же роль).

3. Формализация понятия роли

Этот раздел посвящен уточнению понятия роли для последующего использования этого понятия в моделях. Сначала мы рассмотрим свойства ролей, рассматривающиеся в литературе, а затем обозначим контекст использования нами этого понятия в сценариях взаимодействий.

3.1. Свойства ролей

В литературе понятие роли используется преимущественно в контексте моделей данных и в архитектурных моделях. В таких работах исследуются средства структурной композиции свойств объекта из свойств его ролей; нас же будет интересовать, прежде всего, объединение функциональности, соответствующей поведению в различных ролях. Тем не менее, сами свойства ролей, как в структурном, так и в поведенческом аспектах сходны. Основываясь на обзоре, произведенном в [5], отметим следующие свойства ролей:

- констектуальность: роли имеют смысл только в контексте некоторого взаимодействия;

- объекто-зависимость: экземпляры роли не существуют без объектов;
- множественность: объект может играть несколько ролей одновременно (в частности, несколько экземпляров одной и той же роли);
- иерархичность: роли могут образовывать иерархии ролей.

Эти свойства ролей прослеживаются и в сценариях.

Подчеркнем различие между классами и ролями:

- *классы* задают свойства отдельных объектов;
- *роли* задают позицию и *ответственность* (*responsibility*) объекта в рамках некоторой системы или подсистемы.

Удобно различать понятие роли и экземпляра роли – по аналогии с различением понятий класса и *экземпляра класса*. Под существованием во время выполнения (*run-time*) экземпляра роли мы будем понимать факт выполнения экземпляром агента или компонента системы данной роли в процессе некоторого взаимодействия.

В процессе функционирования системы каждый из ее объектов *выполняет* (или *играет*) некоторую роль (возможно, несколько ролей одновременно); таким образом, можно говорить о существовании экземпляров ролей (одного или нескольких, если объект выполняет сразу несколько ролей одновременно), соответствующих объекту в каждый момент времени. В случае отсутствия явного выделения роли объекта можно говорить о выполнении им некоторой анонимной роли.

Метамодель, иллюстрирующая эти соотношения, может быть описана диаграммой, представленной на Рис. 4.

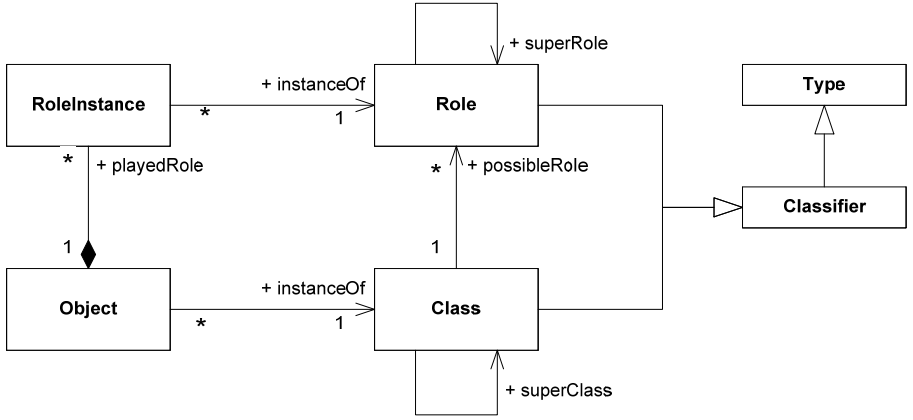


Рис. 4. Метамодель, описывающая соотношения между ролями, классами и их экземплярами

Абстракция роли достаточно исследована с точки зрения ее структурной реализации, например, модели данных, использующие роли [6], связь понятия роли и интерфейса [7], шаблоны реализации ролей [8]; однако с точки зрения объединения поведения в различных сценариях роли практически не исследуются.

Произведем краткий обзор шаблонов для представления ролей, рассматриваемых в работе [8]:

- объединение в *единый тип для ролей* (*single role type*) – все особенности каждой из ролей объединяются в один общий тип;
- использование *отдельного типа для каждой роли* (*separate role type*) – каждая роль трактуется как отдельный тип;
- использование *ролевого объекта* (*role object*) – особенности, присущие роли, объединяются в специальный объект; основной объект является *хостом* (*host object*), объединяющим несколько таких ролевых объектов; при обращениях внешних объектов он использует нужный из ролевых объектов в зависимости от контекста;
- использование *ролевого отношения* (*role relationship*) – специальный объект, объединяющий особенности данной роли, моделирует связь основного объекта в данной роли с некоторым внешним объектом.

Структурировать описание поведения можно различным образом; при рассмотрении поведений, соответствующих различным ролям в рамках некоторой архитектурной ролевой модели, можно достичь соответствия между ролями в их структурном и поведенческом понимании.

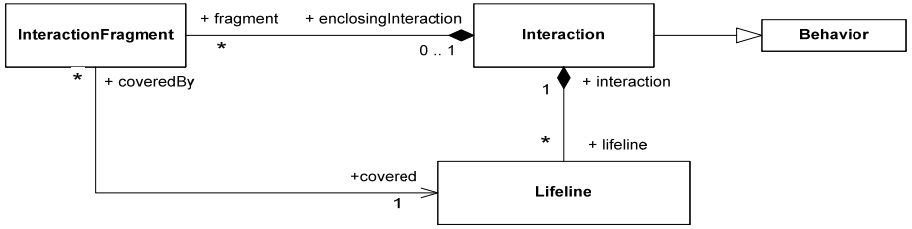


Рис. 5. Фрагмент упрощенной модели взаимодействий UML

3.2. Роли в сценариях взаимодействия

Говоря об описании взаимодействий в системе, можно предложить следующее определение роли: *роль* – это абстракция агента или компонента системы в рамках его ответственностей в некотором взаимодействии. По сути, в результате такого абстрагирования, мы получаем некоторый срез поведения

объекта. Ответственности участников взаимодействия определяются логикой описываемого сценария.

В модели взаимодействий UML при описании некоторого сценария каждой роли естественно сопоставлять оси с одинаковым именем на различных диаграммах. В контексте некоторого упрощения метамодели взаимодействий UML (Рис. 5) взаимосвязь ролей с моделью взаимодействий можно проиллюстрировать диаграммой, представленной на Рис. 6.

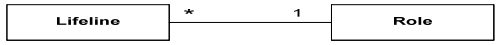


Рис. 6. Роли в метамодели взаимодействий

3.3. Взаимосвязь с аспектами

Использование ролей близко соотносится с аспектно-ориентированным программированием. Под аспектом понимается некоторый срез системы, который объединяет функциональность ее различных частей, соответствующую некоторому классу задач. Таким образом, отчетливо видна связь между описанием сценариев системы и ее аспектами поведения, так как по существу каждый сценарий является описанием какого-либо аспекта системы; при этом каждому аспекту соответствует некоторый набор ролей – эта связь отражена на диаграмме, представленной на Рис. 7.

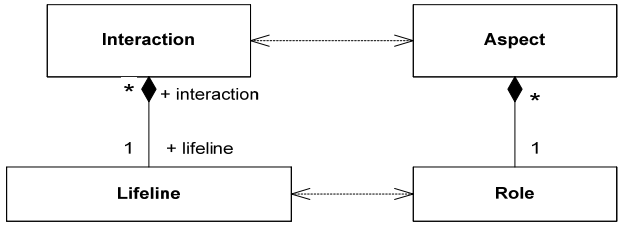


Рис. 7. Взаимосвязь ролей и аспектов с взаимодействиями UML

Исследования взаимосвязей в использовании ролей и аспектов можно найти, например, в [9] и [10].

4. Композиция поведения в разных ролях

В данном разделе будут рассматриваться подходы, позволяющие решать задачу описания построения композиции поведения в различных ролях. Для этого будут рассмотрены некоторые расширения семантики диаграмм

взаимодействия. Прежде чем перейти к рассмотрению расширений, уточним семантику обзорных диаграмм взаимодействия.

4.1. Уточнение семантики композиции

Каждая диаграмма последовательностей естественным образом предполагает наличие постоянного контекста описываемого взаимодействия, т.е. в ней описывается своего рода транзакция взаимодействия объектов. Набор осей задает классы объектов и их возможные роли, которые они могут выполнять.

В дальнейшем под обзорными *диаграммами взаимодействия*, мы понимаем соответствующие диаграммы UML (UML *Interactions Overview Diagrams*), им родственна нотация *высокоуровневых диаграмм* MSC (*High-level* MSC, *HMSC*).

Обзорные диаграммы взаимодействия декларируются в спецификации UML как вариант диаграмм активностей, однако же это отражено лишь в нотации.

Спецификация UML определяет семантику этих диаграмм декларативно; для целей сравнения поведений имеет смысл определить их исполняемую семантику – как вариант семантики диаграмм активности. Семантика же диаграмм активностей определяется в UML в стиле семантики сетей Петри, т.е. через поток *маркеров* (*tokens*) по графу, состоящему из мест, переходов и дуг, их соединяющих. В рассматриваемом случае этим местам соответствуют активности, каждая из которых описывается диаграммой взаимодействия, а переходам соответствуют условные и параллельные ветвления потока маркеров.

Для определения поведения сопоставим каждому объекту набор маркеров. Каждому маркеру ставится в соответствие экземпляр некоторой роли, выполняемой объектом; понятия маркера и экземпляра роли, таким образом, можно в определенном смысле отождествлять. Нахождение маркера для экземпляра роли в некоторой активности, означает, что объект, которому соответствует этот экземпляр роли, участвует во взаимодействии, описываемом сопоставленной этой активности диаграммой взаимодействия. Состояние объекта определяется набором ролей, которые он исполняет, т.е. соответ-ствующим множеством экземпляров ролей и их состояниями. Состояние всей системы определяется текущей разметкой диаграмм маркерами.

Возможность активации перехода для маркера означает, что для объекта, соответствующего этому маркеру одновременно:

- в активности, в которой он находится, выполнены все взаимодействия;
- в диаграмме, в которую ведет переход, выполнены предусловия, задаваемые первым событием, находящимся на оси, которая соответствует данному объекту.

Предусловие зависит от вида такого события следующим образом:

- для проверки условного выражения предусловие перехода соответствует этому выражению;

- для приема сообщения предусловие заключается в наличии такого сообщения во входной очереди сообщений объекта;
- для других видов событий (так называемых, активных событий) предусловие соответствует тождественно истинному выражению, и переход может быть произведен (активирован) в любом случае.

При активации перехода происходит перемещение маркера: маркер, соответствующий данному объекту, убирается из активности, в которой он находился, а в активность, в которую ведет сработавший переход, помещается маркер, соответствующий той роли, которую объект будет в ней выполнять. В случае перехода с распараллеливанием маркеры помещаются во все активности, в которые ведут ветви перехода. Для перехода, объединяющего параллельные ветви, условиями его срабатывания являются наличия маркеров, соответствующих одному и тому же объекту.

Отметим, что такое описание композиции поведения объектов сходно с использованием сетей Петри для описания композиции ролей, предложенном в [11], где рассматривалась программная среда для разработки приложений, основанных на агентной архитектуре.

Рассмотрим теперь следующие методы описания композиции поведения в различных ролях:

- использование специальных меток для «склейки» поведения;
- использование параметризуемых по осям диаграмм;
- задание потоков объектов на обзорных диаграммах взаимодействий.

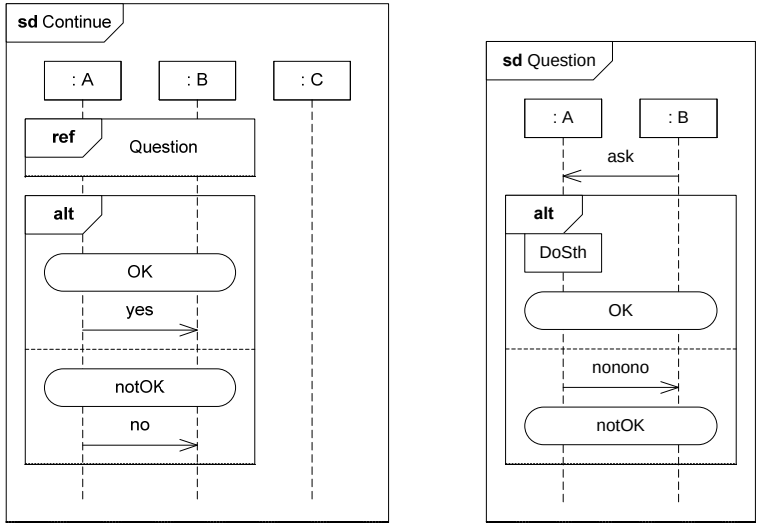


Рис. 8. Пример из спецификации UML: композиция с помощью конструкции «продолжения»

4.2. Композиция через конструкции «продолжения»

Спецификация UML предлагает дополнительный способ композиции поведения через конструкцию продолжения (*continuation*) осей. Семантика таких конструкций носит характер «склейки» поведения по меткам: после фрагмента взаимодействий, заканчивающегося конструкцией продолжения с именем “X”, допустимо продолжение по тем ветвям следующего фрагмента взаимодействий, которые начинаются с конструкции продолжения с тем же именем “X”. Это может быть проиллюстрировано примером, взятым из спецификации UML: диаграмма *Continue* со ссылкой на диаграмму *Question* (Рис. 8) эквивалентна диаграмме, представленной на Рис. 9.

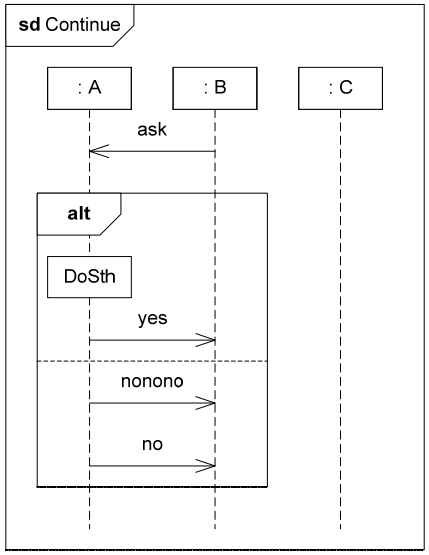


Рис. 9. Пример из спецификации UML: результат композиции с помощью конструкции «продолжения»

Для целей композиции поведения объектов в различных ролях разрешим использование конструкции продолжения не только в варианте, когда она покрывает все оси объектов, участвующих во фрагменте взаимодействия. Кроме того, дополнительно будем использовать следующее правило композиции: на диаграмме, следующей за данной диаграммой, в качестве возможных осей, которые соответствуют продолжению описания поведения, задаваемого осью на данной диаграмме взаимодействий (следование задается переходом между диаграммами на обзорной диаграмме), рассматриваются оси с тем же типом, роль же может быть другой. Естественно, что в качестве продолжения при исполнении выбирается лишь одна ось, в соответствии с выполнением предусловий для данного объекта.

Тогда можно рассмотреть спецификацию системы обмена данными, представленную на Рис. 10, которая будет эквивалентна исходной (Рис. 2), в предположении, что объекты `A` и `B` в исходном случае – это узлы некоторого одного типа *Node*.

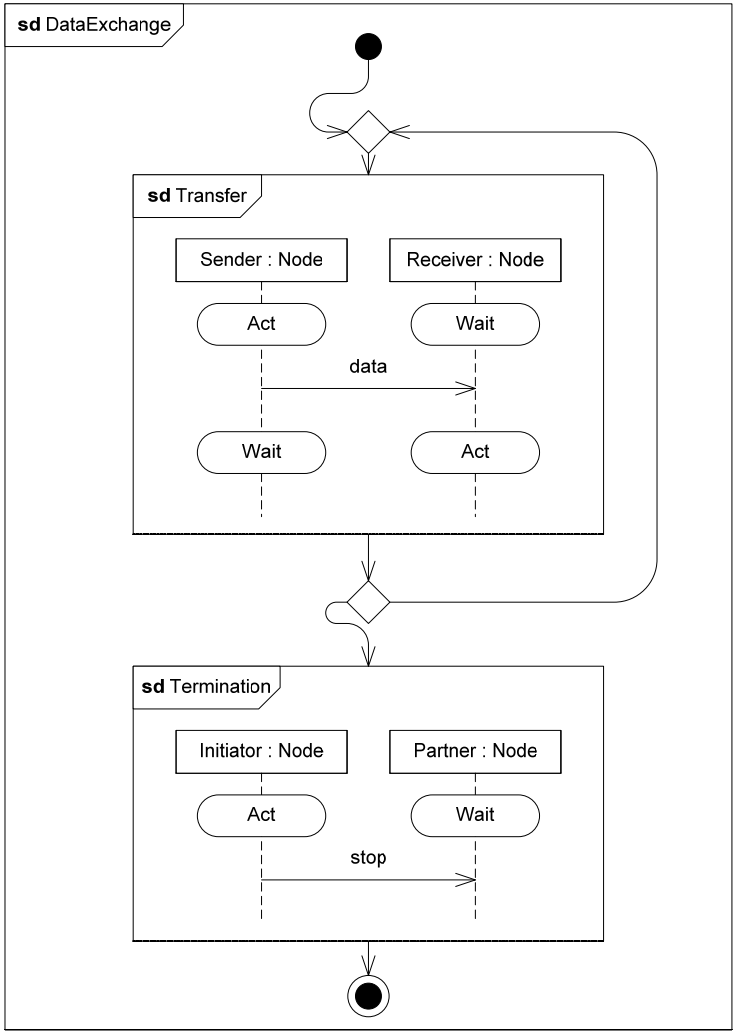


Рис. 10. Спецификация модельной системы обмена данными с помощью конструкций «продолжения»

Данная спецификация задает поведение объектов типа *Node*. Каждый вариант взаимодействия в этой спецификации описан один раз для своего набора ролей, однако несколько утеряна наглядность, присущая диаграммам

взаимодействия, вследствие необходимости отслеживать имена меток конструкций продолжения.

Следует отметить, что в этой спецификации есть некоторая неопределенность, связанная с тем, что в самом начале взаимодействия не задан способ выбора роли (*Sender* или *Receiver*) объектом типа Node. Эта неопределенность может приводить в данном случае к тупиковым ситуациям в функционировании системы, когда объекты совместно начинают выполнять роль *Receiver*, в результате чего бесконечно ожидают приема сигнала.

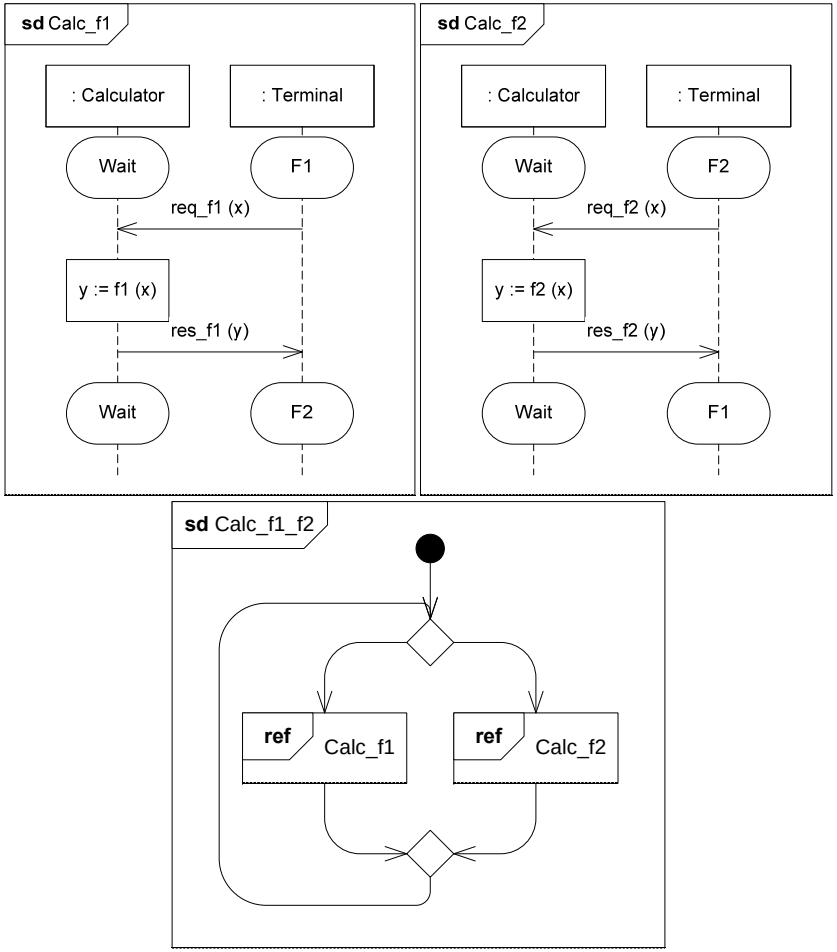


Рис. 11. Спецификация системы вычислителя с несколькими терминалами

Такое расширение композиции через конструкции продолжения дополнительно обеспечивает следующую заслуживающую внимания возможность. Рассмотрим систему, состоящую из вычислителя (*Calculator*), который может вычислять функции `f1` и `f2` и терминалов (*Terminal*), обращающихся к нему за вычислением этих функций. Пусть терминал должен запрашивать вычисление функций, чередуя `f1` и `f2`, а вычислитель должен обладать возможностью производить вычисления функций в любом порядке. Поведение этой системы может быть описано с помощью диаграмм, представленных на Рис. 11.

Тогда в соответствии с правилом композиции описанное здесь поведение для терминала может быть проиллюстрировано обзорной диаграммой взаимодействий, изображенной на Рис. 12.

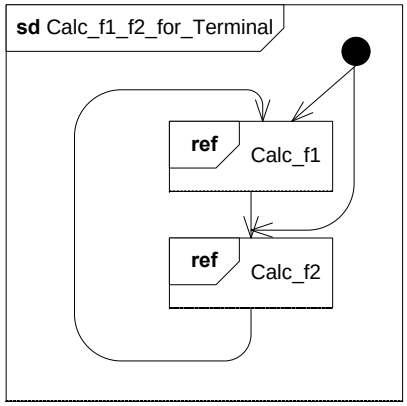


Рис. 12. Поведение вычислителя

В рассмотренном варианте задания композиции элементы, собственно специфицирующие эту композицию, оказываются инкапсулированными внутрь описания отдельного взаимодействия. Это приводит к снижению наглядности описания системы.

Помимо этого, использование такой нотации привносит некоторую автоматоподобную семантику описания, причем в рамках каждого отдельного объекта системы, что не слишком соответствует логике нотации моделей взаимодействий и увеличивает вероятность создания внутренне противоречивых моделей.

4.3. Композиция через параметризацию диаграмм по осям

Рассмотрим альтернативный вариант композиции, в котором используется параметризация диаграмм.

Спецификация UML не говорит явно о возможности параметризации имен осей, однако заметим, что стандарт MSC позволяет задавать имена осей в качестве параметров MSC-диаграммы.

Будем считать параметризацию осей возможной. Тогда, используя ее, рассматриваемую модель обмена данными можно описать с помощью диаграмм, представленных на Рис. 13.

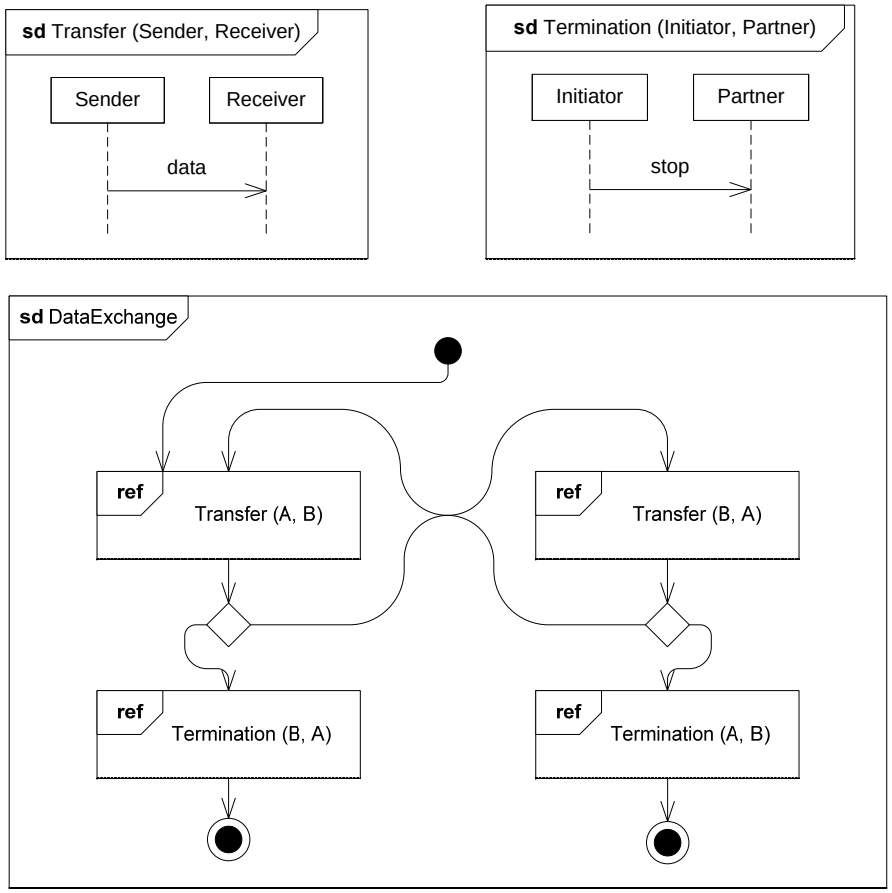


Рис. 13. Спецификация модельной системы обмена данными с помощью параметризации диаграмм по осям

Можно рассматривать статическую и динамическую семантику параметризации. Для статического случая результат соответствует макроподстановке диаграммы вместо ссылки на нее; динамическому случаю соответствует вызов поведения некоторой подсистемы. Связывание фактического параметра, соответствующего некоторому объекту определенного типа, и формального параметра, соответствующего роли в данном взаимодействии, означает *вовлечение* (*acquirement*) этого объекта в данную роль. В зависимости от семантики параметризации можно говорить либо о статическом «обладании» объектом своими ролями, либо о

динамическом «принятии» в роли; это согласуется с различиями архитектурного моделирования ролей.

Заметим, что в данном случае пришлось продублировать описания ссылок на описания протоколов; вместо этого можно попытаться явно описывать задающие значения параметров потоков объектов между диаграммами.

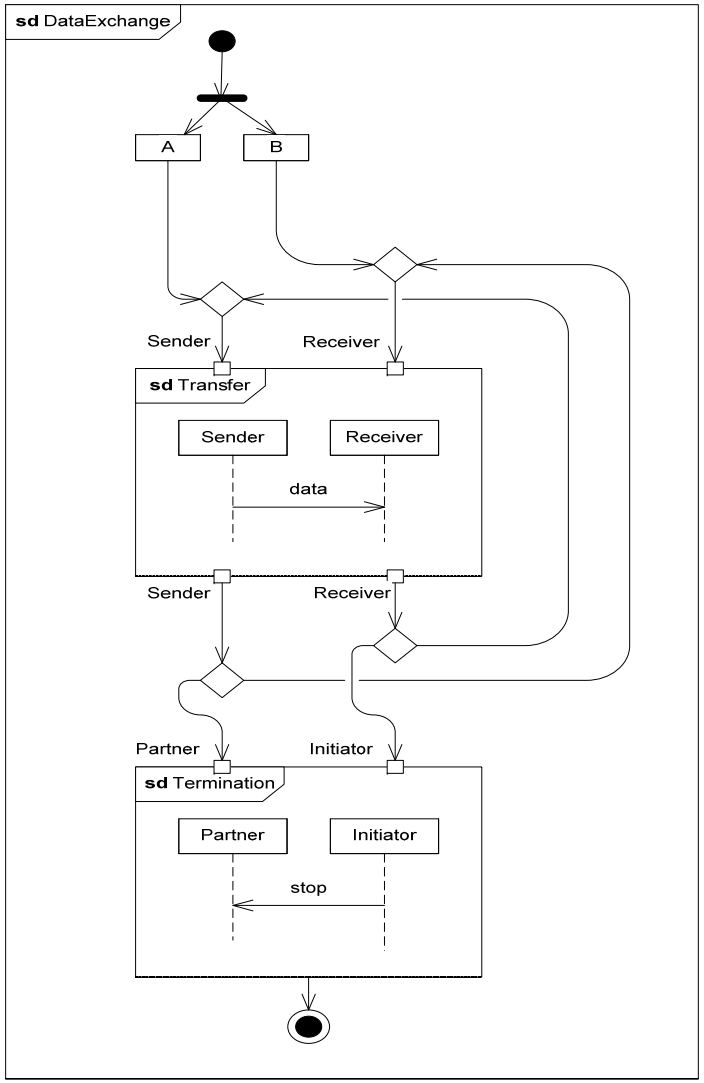


Рис. 14. Спецификация модельной системы обмена данными с помощью явного задания потоков объектов

4.4. Композиция через задание потоков объектов

Снимем некоторые ограничения обзорных диаграмм взаимодействия, приблизив их к диаграммам активности. Рассмотрим возможность ограничения на потоки объектов между диаграммами путем задания их типа. Для этого мы будем использовать символы объектов на дугах диаграмм. Мы будем считать, что такой переход может сработать для маркера, если он поставлен в соответствие объекту, имеющему тип, который соответствует типу, указанному на этом переходе (в качестве вариантов можно рассматривать как строгое соответствие, так и соответствие какому-либо более общему типу). Для переходов с отсутствием такой спецификации предполагается, что они могут срабатывать для маркеров любого типа.

По смыслу, такое описание (рис. 14) сходно с параметризацией, но с явным указанием входных и выходных потоков фактических параметров. Применение таких потоков позволило избежать дублирования ссылок, но при этом сделало более сложным само описание композиции.

5. Отображение в другие модели

Необходимость композиции поведения объектов в различных ролях приводит к еще одному источнику нежелательного поведения системы, не отраженного в сценарной модели; мы столкнулись с одним из таких примеров в предыдущем разделе. Как говорилось во введении, аппроксимация поведения модели взаимодействия другими моделями может помочь выявить пробелы в понимании требований к системе. Ниже мы рассмотрим возможность отображения модели последовательностей в две другие модели UML, позволяющие описывать поведение системы – это автоматные модели (машины состояний) и активности.

Модель машины состояний хорошо подходит для описания поведения отдельного объекта. Для описания систем, состоящих из некоторого набора взаимодействующих объектов, используются модели, представляющих собой набор таких машин состояний, каждая их которых дает описание поведение отдельного объекта системы. При этом предполагается, что существует некоторое окружение (среда поддержки), которое отвечает за передачу сообщений между экземплярами таких машин. Это окружение относится уже к платформо-зависимому уровню системы, сами же машины достаточно просто отображаются в код, поэтому такие модели широко используются для прототипирования. С другой стороны, в таких моделях утеряны явные связи между отдельными объектами системы, модели же активностей, моделирующие как потоки управления, так и потоки данных в системе, позволяют исследовать поведение системы не покомпонентно, а в целом. Поэтому интерес представляет отображение в обе эти модели.

5.1. Отображение в модель машин состояний

Для спецификации поведения объектов системы применяются подходы, основанные на событийных автоматах; в таком случае поведение каждого компонента системы задается расширенным конечным автоматом. Событийные автоматы довольно легко отображаются в код на целевых языках; кроме того, существуют такие распространенные нотации как SDL [12] и машины состояний UML, в которых используется семантика автоматов.

Под *машинами состояний* (*State Machines*) мы понимаем расширенные конечные автоматы в алфавите событий, возможных в системе. Для этих автоматов должны выполняться некоторые дополнительные требования, которые заключаются в выделении некоторых типов состояний автомата, различающихся видами событий, по которым возможны переходы из этих состояний. Эти виды следующие:

- активные состояния, из которых возможен только один переход, происходящий по активному событию (выполнение действия (*Action*), посылка сигнала и т.д.);
- состояния проверки условия, из которых возможны переходы по проверке условий;
- состояния ожидания получения сообщения; из них возможны переходы по событиям приема сообщений.

Состояния ожидания автоматов соответствуют собственно состояниям машин, остальные состояния автоматов соответствуют так называемым псевдосостояниям.

5.1.1. Алгоритм построения машин состояний по диаграммам последовательностей

Сделаем обзор алгоритма построения машины состояния, работающего в случае анонимных ролей:

- синтаксический разбор (*parsing*);
- построение графов, соответствующих диаграммам (*ordering/linking*);
- объединение поведений, заданных диаграмма (*placement/integration*);
- построение срезов (*slicing*);
- построение автоматов, их преобразование в детерминированные автоматы, трансформация в машины состояний (*automata generation & minimization*);
- кодогенерация (*codegeneration*).

Кратко опишем эти фазы.

Фазы синтаксического разбора и построения графов диаграмм зависят от синтаксиса их записи. В результате для каждой диаграммы обзора взаимодействий строится внутреннее представление графа, который состоит из диаграмм и связей между ними, задающих их последовательность. Для диаграмм последовательностей строятся события (и так называемые

псевдособытия) и связи, задающие их последовательность на данной оси. Для inline-конструкций и ссылок на другие диаграммы строятся псевдособытия. Также строятся связи между множествами событий (и псевдособытий) на разных осях, если они взаимосвязаны – т.е. для пар событий посылки-приема сообщения, псевдособытий, соответствующих началам и концам ссылок на другие диаграммы, началам и окончаниям inline-конструкций.

Для объединения поведений в соответствии с графом использования (через ссылки) диаграммами друг друга выбирается диаграмма самого верхнего уровня (считается недопустимым случай рекурсивных ссылок диаграмм друг на друга), и, начиная с этой диаграммы, рекурсивно применяется подстановка поведения используемых в ней диаграмм. После этого производится объединение поведений, заданных различными диаграммами: для каждой пары следования фрагментов взаимодействия происходит построение связей между псевдособытием «конца» каждой оси из предшествующего фрагмента с псевдособытием «начала» оси последующего фрагмента.

По срезам модели, построенным для каждого объекта системы взаимодействий, строятся автоматы: каждое событие среза отображается в переход автомата по этому событию, а каждая связь между событиями на одной оси – в состояние автомата. Для псевдособытий строятся пустые переходы.

После этого производится преобразование автоматов в эквивалентные детерминированные автоматы, после чего полученные автоматы структуризируются в соответствии с требованиями машин состояний.

Таким образом, в полученной машине состояний событиям соответствуют те элементы диаграмм взаимодействия, которые носят конструктивную семантику – т.е. посылка и прием сообщения, выполнение действия, порождение объекта. Элементам, семантика которых носит характер ограничений, как, например, конструкции временных ограничений, инварианты, соответствуют некоторые атрибуты событий, задающие условия, невыполнение которых означает динамическую ошибку.

Описанный алгоритм реализован для MSC-диаграмм в синтезаторе Bridge, разработанном в Институте системного программирования РАН совместно с компанией Klocwork. По построенным автоматам может генерироваться модель системы на языке SDL (очень близком к протокольным машинам состояний UML), прототип системы на языках Си/C++ или Java, тесты для системы на языках TTCN и PPL.

Для рассмотренного в начале примера спецификации системы обмена данными будут сгенерированы (с точностью до имен состояний) автоматы, изображенные в форме машин состояний на Рис. 15.

Степень аппроксимации исходных сценариев автоматными моделями, построенными таким образом, обсуждается в работе [13].

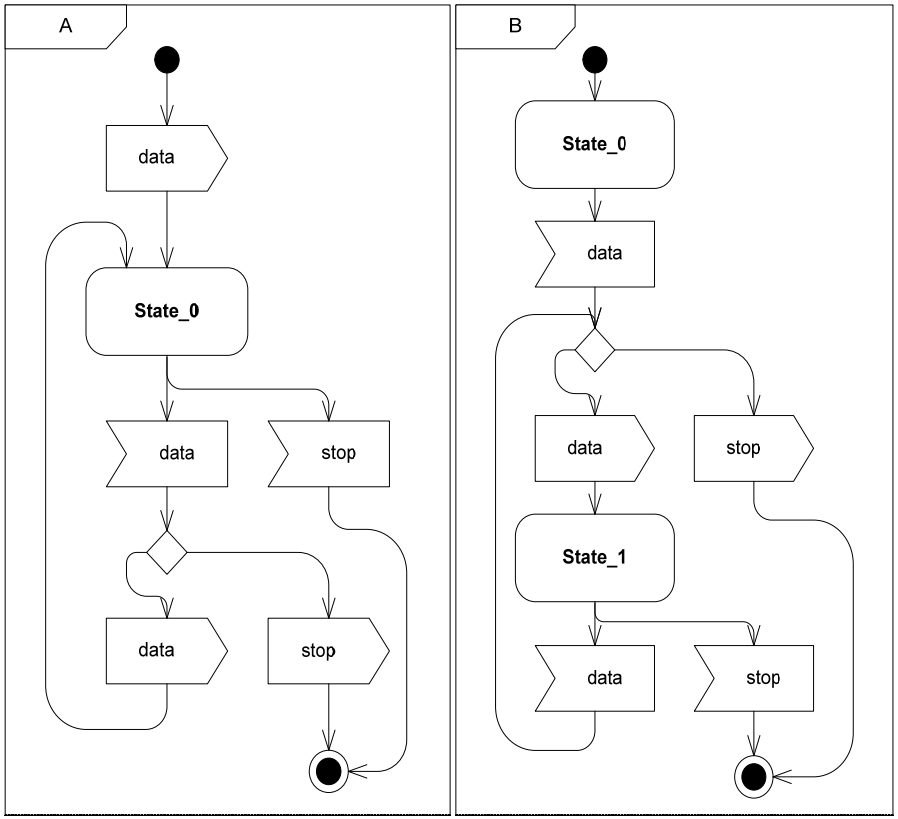


Рис. 15. Машины состояний для объектов модельной системы обмена данными.

5.1.2. Расширение алгоритма синтеза при использовании средств для композиции ролей

В настоящее время для приведенного алгоритма в синтезаторе Bridge реализовано расширение, заключающееся в возможности использования параметризованных ссылок; в том числе, параметризованы могут быть и оси диаграммы. Для этого на фазе интеграции поведений в момент подстановки копии диаграммы, на которую идет ссылка, производится замена формальных параметров на фактические; по существу, это соответствует выполнению параметризованной макроподстановки.

Для возможности построения композиции с помощью конструкций продолжения требуется некоторое расширение алгоритма, позволяющее производить такую «склейку» осей.

Рассматриваемые расширения алгоритма синтеза касаются фаз подстановки и построения срезов.

5.1.3. Структуризация поведения по ролям

Построение структурированной модели машин состояний делает их более масштабируемыми, что расширяет возможности дальнейшего применения получаемых моделей. Наличие взаимосвязи между структурой этой модели и структурой исходной модели взаимодействий дает возможность более тесно сочетать использование обеих этих моделей при описании поведения системы, что позволяет рассматривать поведение системы с разных точек зрения.

Далее мы рассматриваем вариант расширения алгоритма синтеза, заключающийся в структуризации поведения по ролям, т.е. генерации иерархической структуры, состоящей из машин состояний, в которых происходит вызов других машин состояний (т.е. подмашин) в соответствии с использованием ролей. Для этого вместо подстановки поведения, соответствующего роли, для каждой роли генерируется конструкция вызова подмашины, которая состоит из:

- переходов, соответствующих событиям, которые происходят в данной роли;
- состояний, все переходы в которые происходят по событиям, происходящим в данной роли.

Для рассматриваемого примера обмена данными такие подмашины весьма тривиальны (Рис. 16).

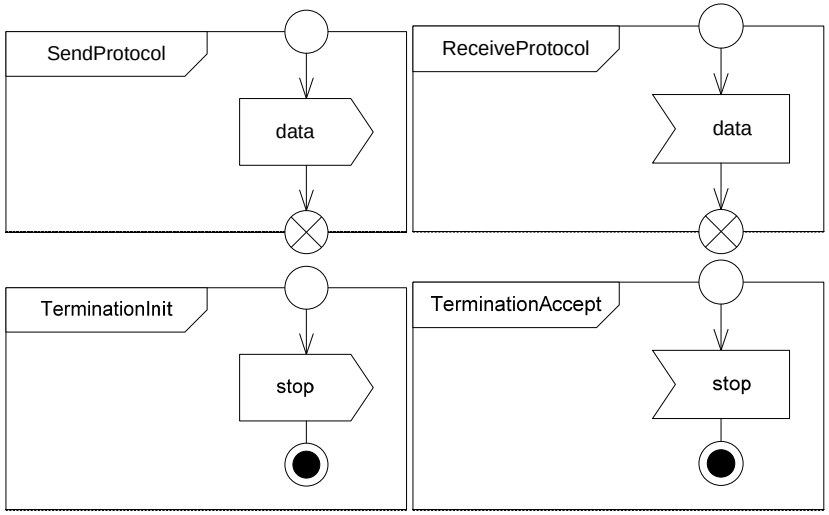


Рис. 16. Подмашины состояний, соответствующие поведению объектов в различных ролях

Другими словами, каждая подмашина состоит просто из отрезка перехода между состояниями по событию принятия или посылки соответствующего сигнала. В данном случае производить такую инкапсуляцию не слишком целесообразно. Тем не менее, если протокол пересылки носит более сложную форму, то такая группировка может быть гораздо более значимой.

Конкретное представление такой иерархии можно рассмотреть в контексте задачи генерации прототипа архитектурной модели системы. Существуют различные способы структурной реализации ролей. Тривиальный способ заключается в простой комбинации всех поведений; простая подстановка именно этому способу и соответствует.

В качестве менее тривиального способа структурной реализации ролей можно рассмотреть шаблон, соответствующий использованию ролевого объекта в работе [5] (Рис. 17).

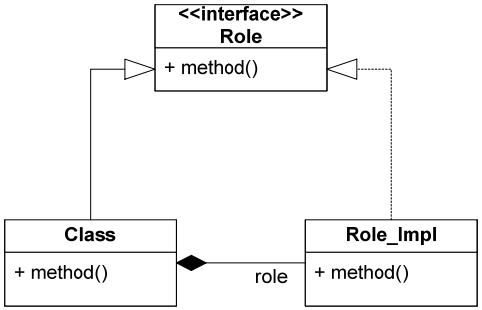


Рис. 17. Шаблон «ролевой объект» для представления ролей в структуре системы

Если в системе существует поведение, связанное с некоторой ролью, то для такой роли заводится интерфейс *Role* и реализация этого поведения – *Role_Impl*. Если объекты типа *Class* могут выполнять эту роль, то такой класс должен быть унаследован от интерфейса *Role*, и он должен агрегировать объект типа *Role_Impl*, который по своей сути соответствует экземпляру роли. Поведение же, соответствующее данной роли, должно делегироваться в *Class* из *Role_Impl*.

Для примера с передачей данных, если считать типы объектов `A` и `B` разными, структура системы будет описываться диаграммой на Рис. 18.

Этот пример показывает, что по описанию взаимодействий может быть автоматически построен и прототип архитектурной модели системы, структура которого соответствует логике исходных сценариев.

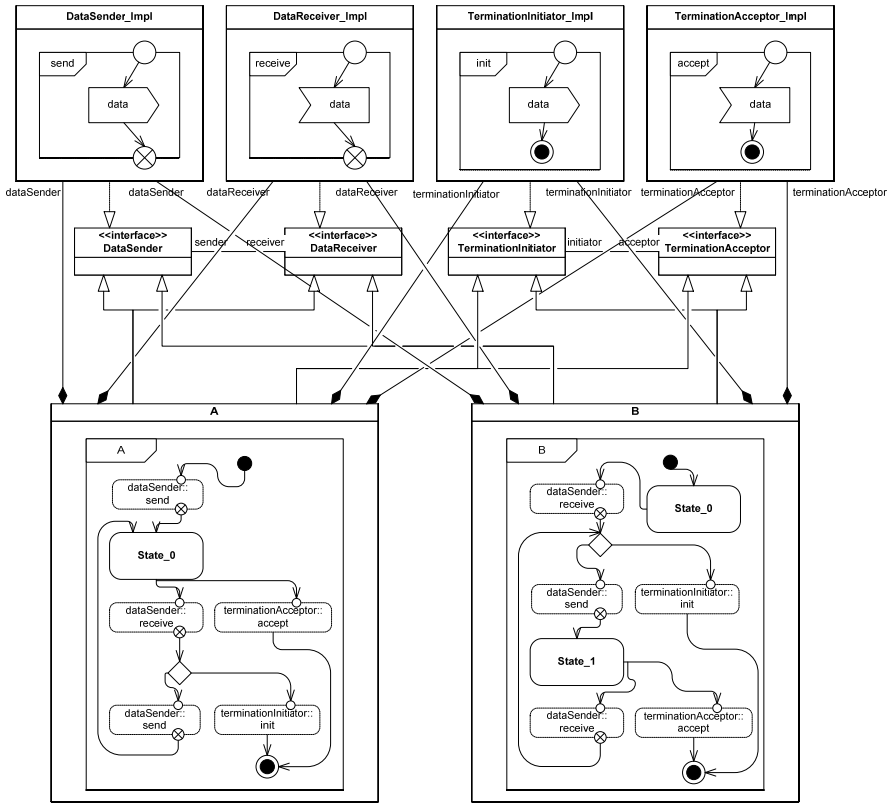


Рис. 18. Машины состояний для объектов модельной системы обмена данными, структурированные по ролям

5.2. Отображение в модель активностей

Диаграммы активностей, используя семантику, сходную с семантикой сетей Петри, хорошо описывают работу системы в целом, так как явным образом моделируют и передачу сообщений. Построение моделей активностей UML по сценариям, описанным с помощью моделей взаимодействий, могло бы позволить производить статическую проверку требований к системе.

Соображения о возможности построения отображения из моделей взаимодействий UML в модели активностей основываются на возможности использования для обзорных диаграмм взаимодействия семантики в стиле сетей Петри, а также возможности отображения отдельных взаимодействий в активности, что продемонстрировано на Рис. 19.

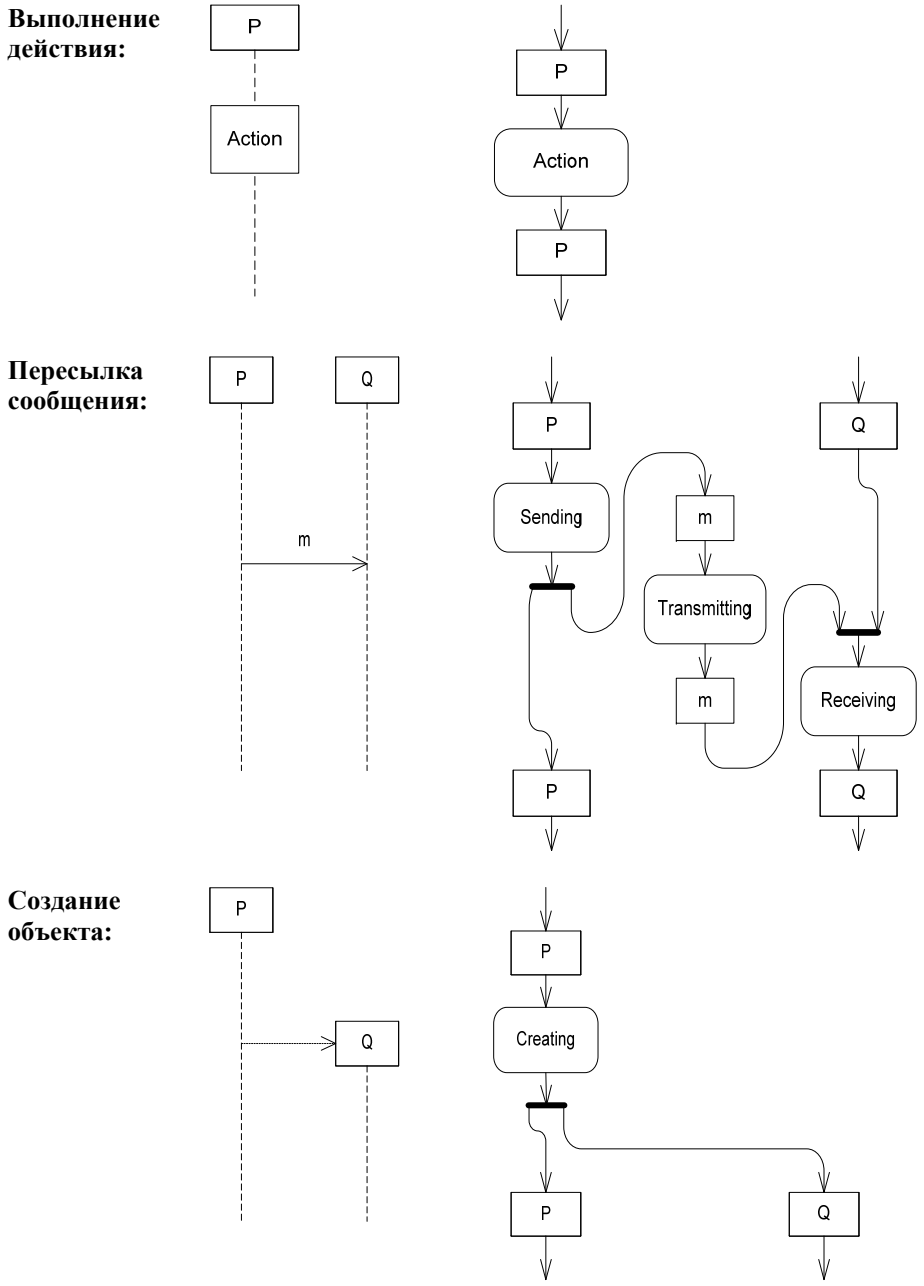


Рис. 19. Отображение взаимодействий в активности

Отметим, что существуют работы, в которых рассматриваемся формализация исполняемой семантики взаимодействий через сети Петри – см. [14].

Результат фазы интеграции в алгоритме построения машин состояний, описанном в предыдущем разделе, практически соответствует диаграмме активности и может быть отображен в нее.

Можно рассматривать вопрос и о структурировании диаграмм активностей в соответствии с имеющимися сценариями и общими в них ролями.

Также возможно отображение машин состояний в диаграммы активностей [15]. Можно сделать предположение, что существо различия между моделью активностей, построенной по UML-взаимодействиям, и моделью активностей, построенной по машинам состояний, которые сгенерированы их тех же UML-взаимодействий, заключается в том, что моделирование передачи сообщений в первом случае происходит отдельными потоками сообщений между активностями посылки и приема, а во втором представляет собой некоторый общий поток, соответствующий каналу передачи. В этом случае модель активностей потенциально может являться некоторым базисом для сравнения поведений.

6. Заключение

В статье было продемонстрировано, что, в отличие от традиционной интерпретации, каждое описание взаимодействий может трактоваться как ориентированное не на собственно объекты, а на роли, выполняемые ими в данном взаимодействии. Для более широкой применимости моделей взаимодействий и более формального их использования можно использовать рассмотренный аппарат для оперирования этими ролями.

В данной работе показано, как можно адаптировать имеющуюся нотацию UML для описания ролей, прежде всего, в плане задания их композиции. Для этого было рассмотрено использование конструкций продолжения и расширение возможностей параметризации UML-взаимодействий, однако у каждого из этих способов имеются свои недостатки. Следует отметить, что композиция через конструкции продолжения ближе к машинам состояний, так как семантика этих конструкций близка семантике к переходам на метку; композиция же через параметризацию – ближе по смыслу к семантике UML-активностей, так как она тем или иным образом задает потоки объектов.

В качестве модели для абстрактного выполнения были рассмотрены машины состояний UML, которые удобны для целей симуляции и генерации прототипа компонентов системы. Было рассмотрено расширение алгоритма синтеза машин состояний по UML-взаимодействиям на случай использования композиции различных ролей, и указаны те из этих расширений, которые поддерживаются разрабатываемым инструментом Bridge, позволяющим автоматически производить подобный синтез. Кроме этого, была обозначена возможность отображения модели UML-взаимодействий в модель

активностей, которая удобна для анализа поведения системы в целом. Анализ моделей, поведение которых дает аппроксимацию поведения, задаваемого описаниями требований к системе в виде моделей взаимодействий, обеспечивает возможность выявлять пробелы в понимании требований к системе.

Другим результатом экспериментов с новой нотацией является вывод о том, что при композиции различных ролей, как и вообще при композиции сценариев, могут возникать неопределенности. В статье был приведен типичный пример возникновения неопределенности с выбором роли, в результате которого у системы может существовать поведение, приводящее к тупиковой ситуации.

Еще одним следствием использования ролей при использовании для описания сценариев модели взаимодействий является достижение взаимосвязи с аспектно-ориентированным программированием.

В качестве развития данной тематики имеет смысл рассмотреть следующие задачи:

- описание преобразований UML-взаимодействий в UML-машины состояний в терминах метамодели языка UML или некоторого ее упрощения;
- строгая формализация исполняемой семантики композиции UML-взаимодействий через диаграммы обзора взаимодействий;
- исследование ситуаций возникновения неопределенностей, связанных с композицией ролей, на основе расширении алгоритма синтеза машин состояний по взаимодействиям UML;
- исследование возможности генерации модели активностей UML по моделям последовательностей и их взаимосвязь с исходными моделями и с моделями машин состояний, полученными по тем же моделям взаимодействий.

Литературы

1. *UML 2.0 Superstructure Specification FTF.* October 8, 2004, OMG Document: ptc/04-10-02 (convenience document).
2. B. Bollig, M. Leucker, T. Noll. *Generalised Regular MSC Languages.* Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS '02), Grenoble, France, 2002.
3. B. Jonsson, G. Padilla. *An Execution Semantics for MSC-2000.* SDL 2001: Meeting UML, 10th International SDL Forum Copenhagen, Denmark, 2001.
4. ITU Recommendation Z.120 *Message Sequence Charts (MSC-2000).* International Telecommunication Union (ITU), Geneva, 1999.
5. F. Steimann. On the Representation of Roles in Object-Oriented and Conceptual Modelling. Data & Knowledge, Volume 35, Issue 1, October 2000.
6. R. K. Wong, H. L. Chau, F. H. Lochovsky. *A Data Model and Semantics of Objects with Dynamic Roles*. Proceedings of the Thirteenth International Conference on Data Engineering table of contents, 1997

7. F. Steimann. *Role = Interface: a merger of concepts*. Journal of Object-Oriented Programming 14:4, 2001
8. M. Fowler. *Dealing with Roles*. Working Draft (http://martinfowler.com/apsupp/roles.pdf), July 1997.
9. E. A. Kendall. *Aspect-Oriented Programming for Role Models.* Lecture Notes Computer Science, Vol. 1743, 1999.
10. G. Georg, R. B. France. *UML Aspect Specification Using Role Models.* OOIS 2002: Object-Oriented Information Systems, 8th International Conference, Montpellier, France, 2002.
11. M. Becht, T. Gurzki, J. Klarmann, M. Muscholl. *ROPE: Role Oriented Programming Environment for Multiagent Systems.* Proceedings of the 4th IFCIS Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland, September 1999.
12. ITU-T Recommendation Z.100 *System Description and Definition Language (SDL-2000).* International Telecommunication Union (ITU), Geneva, 1999.
13. N. Mansurov, D. Vasura. *Approximation of (H)MSC semantics by Event Automata*. SAM'2000 workshop, Grenoble, France, 2000.
14. St. Heymer. *A Semantics for MSC Based on Petri-Net Components.* SAM 2000, 2nd Workshop on SDL and MSC, Col de Porte, Grenoble, France, 2000.
15. Z. Hu, S. M. Shatz. *Mapping UML Diagrams to a Petri Net Notation for System Simulation.* Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), Banff, Alberta, Canada, 2004.