Р11-2010-145

В. Т. Димитров[1]

ПРОБЛЕМЫ СПЕЦИФИКАЦИИ СЕМАНТИКИ БИЗНЕС-ПРОЦЕССОВ

[1]Софийский университет, София

Димитров В. Т. P11-2010-145
Проблемы спецификации семантики бизнес-процессов

Рассматривается развитие понятия бизнес-процессов из бизнеса в информатике. Особое внимание уделяется этому понятию в информатике в контексте современного бизнеса и требований к программному обеспечению. Спецификация и реализация бизнес-процессов с применением технологий, основывающихся на сервис-ориентированной архитектуре (Service-Oriented Architecture — SOA), имеют ряд специфических проблем. С одной стороны, требования бизнеса определяют быстрейшую реализацию программного обеспечения (ПО) бизнес-процессов. С другой стороны, стандартов, от которых зависит SOA, еще не существует. В результате создаются инструментарии, допускающие противоречия в семантике представляемых бизнес-процессов, не говоря о допуске генерации неработающего кода. В работе представлены основные средства спецификации бизнес-процессов: диаграммы деятельности WS-BPMN, WS-BPEL и UML.

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2010

Dimitrov V. T. P11-2010-145
Specification Problems of Business Process Semantics

In this paper genesis of the term «business process» is discussed: from the business to Informatics. Special attention is given to this term in Informatics in the context of current business development and its requirements on the software. Specification and implementation of business processes with Service-Oriented Architecture (SOA) based technologies have specific issues. Business demands are for fastest implementation of business process software, but SOA standards are still not mature. As a result of that, software tools are realized that permit business process with contradictive semantics to be specified and even to generate non-working code. Here, the main notations for business process specification are analyzed: WS-BPMN, WS-BPEL and UML activity diagrams.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2010

ВВЕДЕНИЕ

В последние несколько лет бизнес и информатику стали переделывать под процессную ориентацию [1, 2]. Это явление связано с неудачами в применении систематического подхода. С ростом возможностей вычислительной техники встал вопрос об объединении в единую систему всех самостоятельных программ, реализующих (или поддерживающих) функции предприятия. Большие расходы на увязки этих программных ресурсов привели к идеям систематического подхода к разработке программного обеспечения (ПО), что привело в свою очередь к разработке автоматизированных систем управления (АСУ) [3]. В АСУ предполагалось, что процессы на предприятиях мало чем отличаются друг от друга. К сожалению, это не так, большие средства были затрачены на создание этих систем, но их внедрение оказалось более трудоемким из-за необходимости многочисленных доработок.

Ситуация усложнилась в 90-х гг. прошлого века, когда мировая экономика из индустриальной превратилась в экономику знаний и услуг [4]. Идея индустриальной экономики заключалась в том, что каждое предприятие является сильносвязанной системой, которая взаимодействует с окружающей средой только рынком. В 1990-х гг. появилось много новых форм сотрудничества между предприятиями и оформилась основная характеристика современного бизнеса — чтобы выжить в условиях быстроменяющегося рынка, каждое предприятие должно быть открытым для всесторонних взаимодействий с другими предприятиями [5]. В этой модели ведения бизнеса самым главным качеством предприятия является гибкость в предоставлении своих конкурентных преимуществ. Из-за этого все виды деятельности, которые не являются конкурентным преимуществом предприятия, выводятся вне предприятия (outsourcing) туда, где они эффективнее реализуются. В конечном итоге получается, что предприятие является уже только звеном в цепи стоимости или цепи поставок, и большая часть функциональности этого звена самостоятельно не реализуется внутри него.

На основе этих новых бизнес-моделей: цепей поставок, цепей стоимости, управления ресурсами предприятия и т. п., повсеместно стала применяться концепция бизнес-процессов. Анализ предприятия ведется на основе бизнес-процессов предприятия и их реализации. Термин «бизнес-процесс» довольно

старый, но ранее он применялся в основном для организации промышленного производства.

Реорганизация бизнеса на основе вышеуказанных бизнес-моделей потребовала ввести концепцию процессов во всех стандартах и в разработке программного обеспечения. Бизнес-процессы уже не рассматриваются как процессы в рамках отдельного предприятия, скорее, предприятия рассматриваются как часть глобальных процессов, реализующих функциональности. Чаще всего, это та функциональность, где предприятие имеет конкурентное преимущество.

Эти изменения в бизнесе привели к тому, что в 90-е гг. прошлого века началась интенсивная разработка адекватного программного обеспечения типа SCM (Supply Chain Management) [6], CRM (Customer Relations Management) [7], ERP (Enterprise Resource Planning) [8] и др.

Программное обеспечение играет ведущую роль в современном бизнесе. С одной стороны, оно позволяет предприятию гибко приспосабливаться к быстроменяющейся бизнес-среде. Насколько гибким является программное обеспечение предприятия — настолько гибок бизнес предприятия. С другой стороны, оказывается, что единственный ресурс, в который можно инвестировать до бесконечности — это программное обеспечение. Для всех других ресурсов инвестиция имеет какой-то уровень, после достижения которого дальнейшие инвестиции невыгодны. Кроме того, инвестиции в программное обеспечение являются одним из основных средств получения конкурентных преимуществ предприятия.

Процессный подход оказался тяжелым бременем как для бизнеса, так и для организации стандартизации, а также для поставщиков ПО. Многие предприятия не смогли адаптироваться к ним и ушли с рынка в 1990-х гг. и начале нового века. Организации стандартизации с некоторым опозданием начали пересматривать все стандарты, связанные с организацией и управлением, чтобы ввести процессный подход. И в настоящее время у многих организаций возникают проблемы с внедрением процессного подхода.

Особенно сложная ситуация сложилась с поставщиками ПО. Объектно-ориентированное программирование в 90-х гг. прошлого века и объектно-ориентированное проектирование в начале нового века только что освоено, а уже появились новые требования к разработке процессно-ориентированного ПО. Это означает переосмысление имеющихся технологий, внедрение процессного подхода к своей работе по требованиям рынка и стандартам, а также разработку новых процессов и технологий разработки процессно-ориентированного ПО.

Первой попыткой был переход от объектно-ориентированного подхода к проектированию и программированию компонент. Ведь только отдельные части ПО реализуются на данном предприятии из глобального бизнес-процесса, а эти части не просто классы — это сильно связанные группы классов с об-

щим интерфейсом компонент. Этот опыт не только поспособствовал развитию объектно-ориентированного подхода, но и решил проблему процессного подхода.

С началом нового века процессный подход стал развиваться в двух направлениях. На высшем бизнес-уровне ведутся работы по стандарту ITIL (IT Infrastructure Library) [9]. Это библиотека лучших из применяемых на практике способов организации работы предприятий, занимающихся предоставлением услуг в области информационных технологий. В семи томах библиотеки описан весь набор процессов, необходимых для обеспечения постоянно высокого качества сервисов и повышения степени удовлетворенности пользователей. Следует отметить, что все эти процессы ориентированы не просто на обеспечение бесперебойной работы компонент инфраструктуры, а в гораздо большей степени на выполнение требований пользователя и заказчика. В конечном счете все процессы ITIL работают на повышение конкурентоспособности, так как в наше время даже внутренние подразделения предприятия не могут чувствовать себя в абсолютной безопасности из-за вынужденной конкуренции с предприятиями аутсорсинга.

Повышению уровня бизнеса способствует применение сервис-ориентированной архитектуры (Service-Oriented Architecture — SOA) из существующей серии стандартов для веб-услуг WS* [10] с использованием стандартов веб-услуг второго поколения (Web Services — WS).

Ожесточенная конкурентная борьба между поставщиками ПО привела к неслыханным до недавнего прошлого слияниям и консорциумам. Эти усилия способствовали разрешению проблем по применениям процессного подхода, но это не означает, что уже существуют полностью готовые технологии. До кризиса ведущие поставщики ПО для разработки сервис-ориентированной архитектуры поставляли новые версии и изменения каждые три месяца, не заботясь о совместимости с предыдущими версиями технологии. Сейчас темпы замедлились, новая версия появляется каждое полугодие. Основной задачей является достижение большей функциональности инструментов для стандартизации своего решения. Применение такого ПО — это как работа на минном поле. Оказалось, что специалистов, способных работать с этим ПО, слишком мало, что привело к созданию новых организационных форм — «летучих бригад» для установки и построения работоспособных сред.

Из этого можно выделить следующие проблемы.

1. Процессная ориентация в ITIL и WS-* (SOA) пока еще не доработана и плохо поддерживается инструментами.

2. Семантика бизнес-процессов, представляемая средствами диаграмм деятельности ITIL, BPMN, WS-BPEL, WS-CDL и UML, является проблемной.

3. Разработка инструментариев поддержки семантики бизнес-процессов является первостепенной задачей поставщиков сервис-ориентированной архитектуры.

1. О НАУЧНЫХ ВЫЧИСЛЕНИЯХ И ГРИД-ТЕХНОЛОГИЯХ

В конце прошлого века идея грида была оформлена в самостоятельной концепции. В [11] представлена идея «компьютинг как сервис» (Computing as a Service). Одновременно авторы представили будущее грида чуть ли не во всех сферах информатики: начиная со взаимодействия с пользователями, требований к безопасности информации и заканчивая возможностями реализации исполнения отдельных задач. Идея грида вдохновила как научный мир, так и бизнес, тем более что предлагался исследовательский прототип Globus Toolkit [12]. Концепция грида не обошла тематику научных вычислений. Обычно новые концепции в информатике делают акцент на решении проблем бизнеса. Редко когда новая концепция берется решать специфические проблемы научных вычислений. Такой проблемой, например, является дистанционное управление экспериментом. Не будем в деталях рассматривать концепции грида.

Все крупные поставщики компьютерной техники и программного обеспечения типа IBM, Oracle, Microsoft и др. занимаются разработкой грида. Они с самого начала принимали участие во всех грид-инициативах типа OGF (Open Grid Forum) [13]. Ранней инициативой IBM было внедрение в бизнесе Globus Toolkit. Со временем оказалось, что грид из [11] скоро не получится — необходимы многие годы на исследование и разработки необходимых стандартов его реализации. Тем более оказалось, что применение Globus Toolkit слишком дорого и не подходит для использования в бизнесе. Это привело к тому, что в 2004 г. в рамках OGF был сформирован Enterprise Grid Alliance [14]. Консорциум поставил себе задачу внедрить грид-технологии на предприятиях в обозримом будущем. В конечном итоге это привело ко многим результатам — начиная с блейд-серверов и заканчивая облачными вычислениями.

Между тем работы по стандартизации грида замедлились. Что же произошло? Главное в грид-компьютинге — это сервис. Эта же идея лежит в основе OGSA (Open Grid Services Architecture) [15]. Таким образом, грид должен быть реализован с применением сервис-ориентированной архитектуры, и, кроме того, ПО, работающее в грид-среде, должно быть сервис-ориентированным. Но, как мы выяснили, SOA — это пока еще недоработанная технология, основанная на веб-сервисах. Все силы были брошены на стандартизации SOA. Как только какая-то спецификация дорабатывается (например, WS-WSDL), происходит ее доработка и на грид. Надо подчеркнуть, что будущее грида определяется сервис-ориентированной архитектурой! Реализация грида должна осуществляться с применением SOA. Программное обеспечение, работающее в гриде, должно быть ориентировано к сервисам. Другими словами, пользователи грида будут компоновать свои приложения в виде процессов из сервисов, а не программировать языками системного уровня. К примеру, функциональность ROOT [16] должна присутствовать в

грид-среде в виде сервисов, а не в виде среды разработки и исполнения типа UCSD Pascal [17] 70-х гг. прошлого века.

Правда, мы пока еще далеки от внедрения сервис-ориентированного грида. Попытки использования этих идей в научных вычислениях (в CMS и ALICE) находятся на начальном этапе, и реальная область их применения еще не определена.

2. СРЕДСТВА ОПИСАНИЯ БИЗНЕС-ПРОЦЕССОВ

Основными средствами описания бизнес-процессов являются BPMN-, UML-диаграммы деятельности, WS-BPEL и WS-CDL. Кроме того, существуют средства частного употребления типа использованных средств в Aris [18], однако у них нет перспективы стать индустриальными стандартами.

В этом пункте будет дана краткая характеристика места и роли каждого из вышеперечисленных средств.

Для описания бизнес-процессов на самом высоком уровне предназначен BPMN. Это графическая нотация, которая хорошо воспринимается конечными пользователями, она довольно четкая для ведения бизнес-анализа. Назначение BPNM определяет способы его применения. Итеративный подход позволяет постепенно наращивать детализацию диаграммы: начиная со схематизации процесса и доводя его до конечного состояния, с которого можно генерировать скелет кода процесса. В общем, все элементы нотации можно подвергнуть итеративной детализации. Кроме того, элементы нотации понятны конечному пользователю, так как они имеют интуитивно ясную семантику, связанную с бизнес-процессами, а не с их реализацией в виде веб-услуг. В ходе детализации можно указать и составляющие реализации отдельных элементов. Например, бизнес процессы в нотации BPMN можно представить в виде композиции задач. Задача — это элементарная единица работы на определенном уровне абстракции. В процессе все задачи должны быть одного уровня абстракции. Каждую задачу можно реализовать в виде процесса, веб-сервиса или просто исполнительного кода этого процесса. Пример BPMN-диаграммы приводится в приложении 1.

У WS-BPEL другая задача: рекурсивно компоновать веб-сервисы в процессы. Сам процесс также является веб-сервисом. Этим языком можно очень детально специфицировать вызов веб-сервисов. Язык WS-BPEL разработан на основе XML. Он неудобен для программирования, из-за чего в инструментах чаще всего представлен в виде графической нотации. Сами процессы представлены в диаграмме, а элементы языка — в графической нотации. Такой подход использует IBM в продукте IBM WebSphere Integration Developer [19]. На этом уровне итерации и детализация не имеют места. Здесь необходимо программирование — написание кода. Чаще всего скелет кода WS-BPEL-

процесса автоматически генерируется спецификацией BPMN. Такой подход, например, применяется в IBM WebSphere Business Modeler [20]. Описание, пример WS-BPEL-кода и диаграмма приводятся в приложении 2.

Диаграммы деятельности UML применяются как средство описания бизнес-процессов. Они имеют хорошо специфицированную семантику метасистемы UML. С другой стороны, диаграммы деятельности имеют двоякое назначение: кроме описания бизнес-процессов на самом высоком уровне они применяются и для описания сложной деятельности классов. Эти диаграммы не связаны с бизнес-анализом и веб-сервисом. В этом смысле BPMN более адекватное средство для описания бизнес-процессов, тем более что в его разработке учитывался опыт использования диаграмм деятельности. Можно сказать, что диаграммы деятельности подходят для проектирования и программирования, а BPMN-диаграммы — для анализа бизнеса.

Сети Петри были использованы при разработке BPMN и диаграмм деятельности. Они были расширены до выразительной мощности машины Тьюринга. Спецификация WS-BPEL использует процессную алгебру типа CSP и π-вычисления. Спецификации BPMN и WS-BPEL не имеют формальной спецификации семантики.

Последним средством, рассматриваемым нами, является WS-CDL. Среди его достоинств — наличие формальной спецификации семантики и утверждение в стандарте, однако он практически не используется. Его назначение — описание сотрудничества веб-сервисов на глобальном уровне. Он не является языком программирования. Например, реализацию композиции веб-сервисов можно проверять на соответствие с WS-CDL-спецификацией, т. е. хореографией. Существуют работы, указывающие на возможность генерации WS-BPEL-кода с WS-CDL, но они не востребованы в отличие от BPMN в WS-BPEL, являющегося естественным путем разработки.

3. СВОЙСТВА БИЗНЕС-ПРОЦЕССОВ

Попробуем систематизировать свойства бизнес-процессов, являющиеся объектом настоящего исследования. Самое общее разделение этих свойств можно сделать на желанные и нежеланные. Надо отметить, что одно свойство может быть желанным для некоторых проектных требований и быть нежеланным для других. Например, существует проектное требование, чтобы система остановилась через некоторое время после завершения работы. Свойство бесконечной работы этой системы является нежеланным. С другой стороны, если систему проектируют на обслуживание непрерывного процесса, то ее остановка является нежеланным свойством, так как система должна работать бесконечно.

Итак, с точки зрения проектирования, системы процессов можно подразделить на системы, которые обслуживают непрерывные процессы и исполняются бесконечно, и такие системы процессов, которые после завершения задания прекращают свою активность. В первом случае имеются следующие возможности:

1) система работает бесконечно и делает полезную работу;

2) система работает бесконечно, но пользы от этого нет;

3) система останавливается согласно обстоятельствам, определенным проектом, — скорее всего, это внешние обстоятельства, от которых зависит система;

4) система останавливается из-за ошибки проектирования.

Во втором случае предполагается, что система процессов будет работать некоторое время и остановится. В этом случае имеются следующие возможности:

1) система окончила свою работу в желанном состоянии, т. е. она успешно выполнила свою работу или прекратила свою активность из-за обстоятельств, не зависящих от нее, но реагировала адекватным образом;

2) система окончила свою работу в нежеланном состоянии; причиной может быть ошибка в проекте, например, блокировка;

3) система не останавливается. Например, система повторяет цикл до бесконечности — это ошибка в проекте. Предполагалось, что система будет работать время, необходимое для выполнения полезной работы, и после этого остановится.

Некоторые из вышеупомянутых ситуаций хорошо сформулированы в формальных системах, а другие не имеют однозначной формулировки — их надо определять конкретно для каждого отдельного проекта.

Ситуации, охваченные системами верификации процессов:

- мертвая блокировка;
- дивергенция;
- детерминизм;
- бесконечное исполнение;
- достижимость;
- условия, описанные формулами линейной темпоральной логики;
- эквивалентность.

Каждую из них рассмотрим отдельно.

Мертвая блокировка — это состояние, когда дальнейшее исполнение невозможно. В этом случае каждый из процессов пробует исполнить какое-то свое действие. Процессы не могут согласовать между собой, какое будет следующее действие (и какого именно процесса), и в результате ничего не происходит. Система блокирована, дальнейшее исполнение не происходит, так как процессы выжидают до бесконечности.

Дивергенция системы — это состояние, когда система участвует до бесконечности в обслуживании внутренних событий и не участвует в полезных внешних событиях. («Живая» блокировка — это тот случай, когда система работает бесконечно, но не делает полезную работу. Это вид дивергенции.) Система в таком состоянии бесполезна. Она использует до бесконечности компьютерные ресурсы, этим она и хуже «мертвой» блокировки. При дивергенции, в сущности, система после некоторого времени начинает работать хаотически.

Детерминистическая система — когда во всех ее состояниях переход к следующему состоянию однозначно определен. Другими словами, нет состояния, в котором некоторое событие может привести к двум разным состояниям, т. е. выбор перехода не определен. Иногда на уровне реализации такая неопределенность может означать ошибку. На уровне спецификации неопределенность такого типа дает возможность разных реализаций, что является более высоким уровнем абстракции.

Бесконечное исполнение системы, как видно из вышеизложенного текста, может быть или не быть ошибкой.

В каждом проекте существуют специфические требования. Проектант должен указать, что является желанным или нежеланным в системе. Это можно сформулировать в виде логических выражений. Спецификацию системы можно было бы проверить на достижимость этих условий.

Желанные и нежеланные условия можно разработать детально, если применить линейную темпоральную логику. Точнее, это можно сделать с применением модальных операторов. В общем, система проверяется на бесконечные последовательности исполнения с некоторой позиции. Эти формулы должны быть истинными. Операторы, которые применяются:

- оΦ, формула Φ должна быть истинной в следующем состоянии;
- □Φ, формула Φ должна быть истинной во всей следующей последовательности;
- ◇Φ, формула Φ эвентуально будет истинной где-то дальше в последовательности;
- ΨUΦ, Φ в рассматриваемой позиции, а Ψ должна быть истинной до нее;
- ΨRΦ, Φ должна быть истинной до первого состояния, в котором Ψ становится истинной. Если Ψ никогда не становится истинной, то тогда Φ должна быть всегда истинной.

С помощью этих формул система точнее исследуется, но подобное исследование нуждается в очень хорошей подготовке проектанта.

И наконец, при разработке спецификации некоторой системы процесс осуществляется итерациями. В результате получается последовательность более детальных спецификаций. В конце этой последовательности находится реализация системы. На каждой итерации надо проверять эквивалентность новой спецификации предыдущей, или является ли текущая спецификация

реализацией предыдущей. Смысл эквивалентности может быть следующим: по следам исполнения, по семантике аварийных ситуаций и по семантике дивергенций. Другими словами:

1) следы исполнения детализированной системы должны быть детализированными следами специфицированной системы;

2) множество нежеланных конечных состояний детализированной системы должно содержать детализированные нежеланные конечные состояния спецификаций;

3) множество дивергенций детализированной системы должно содержать детализированные дивергенции спецификаций.

ЛИТЕРАТУРА

1. Business process orientation. http://en.wikipedia.org/wiki/Business_process_orientation
2. Business process orientation. A blog about business process management and process-oriented organizational design. http://www.processorientation.com/
3. Автоматизированная система управления, http://ru.wikipedia.org/wiki/ Автоматизированная_система_управления
4. Service economy. http://en.wikipedia.org/wiki/Service_economy
5. *Laudon K., Laudon J.* Management Information Systems. 11/E, Prentice Hall, 2010. P. 672.
6. Supply chain management. http://en.wikipedia.org/wiki/Supply_chain_management
7. Customer relationship management. http://en.wikipedia.org/wiki/Customer_relationship_management
8. Enterprise resource planning. http://en.wikipedia.org/wiki/Enterprise_resource_planning
9. ITIL, http://www.itil-officialsite.com
10. Web service, http://en.wikipedia.org/wiki/Web_service
11. The Grid: Blueprint for a New Computing Infrastructure / Eds.: I. Foster, C. Kesselman. Morgan Kaufmann, 1998. P. 675.
12. Globus Toolkit. http://www.globus.org/
13. Open Grid Forum. http://www.gridforum.org/
14. Enterprise Grid Alliance. http://www.ggf.org/gf/docs/egadocs.php
15. Open Grid Services Architecture Basic Execution Service. http://www.ggf.org/documents/GFD.108.pdf
16. ROOT. http://root.cern.ch
17. UCSD Pascal. http://en.wikipedia.org/wiki/UCSD_Pascal
18. IDS Scheer. http://www.ids-scheer.com/international/en
19. WebSphere Integration Developer. http://www-01.ibm.com/software/integration/wid/
20. WebSphere Business Modeler Advanced. http://www-01.ibm.com/software/integration/wbimodeler/advanced/features/

ПРИЛОЖЕНИЕ 1. BPMN

BPMN предназначен для применения следующими категориями бизнес-пользователей:

- бизнес-анализаторами, которые расписывают процесс;
- разработчиками, которые реализуют бизнес-процессы в технологических решениях;
- людьми бизнеса, которые управляют и наблюдают за этими процессами.

При разработке BPMN учитывалась возможность автоматической генерации из спецификации в WS-BPEL исполнимого кода. Таким образом, BPMN является еще мостом между проектированием бизнес-процессов и их реализацией.

Основный элемент BPMN — это диаграмма бизнес-процесса (BPD — Business Process Diagram). На диаграмме операции бизнес-процесса обозначены графическими объектами. Модель бизнес-процесса представлена сетями графических объектов, которые подразделяются на деятельности (работы) и контроль потока управления, определяющего порядок исполнения.

Основная цель разработки BPMN — получение простого механизма разработки бизнес-процессов с возможностью выражения их сложности. Вместе с основными элементами нотации можно применять варианты элементов и добавлять информацию о сложных процессах, не вводя значительных изменений в диаграмму. Имеются четыре категории элементов:

- объекты потока управления,
- связывающие объекты,
- коридоры,
- артефакты.

Объектами потока управления являются

- события,
- деятельности,
- порты.

Событие — это то, что происходит во время исполнения бизнес-процесса. События влияют на ход исполнения, они переключают или дают результат. Имеются три вида событий (рис. 1):

- Start (начало),
- Intermediate (промежуточное),
- End (конец).

Деятельность — это обобщенная единица работы. Она может быть атомарной или составной, т. е. задачей (Task) или процессом (Subprocess) (рис. 2).

Порты используются для направления и слияния потока управления. Порты подразделяются на традиционные блоки решения, развилки, слияния и соединения потоков. Маркеры в блоке указывают на вид порта (рис. 3).

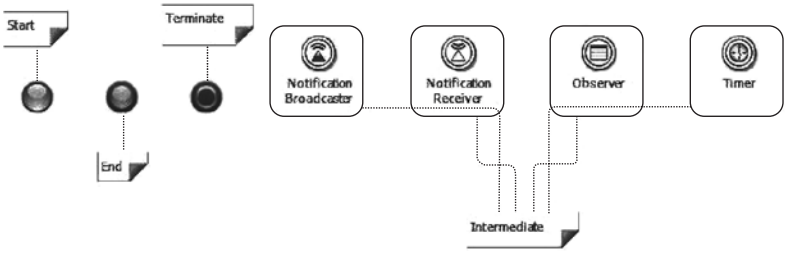

Рис. 1. События

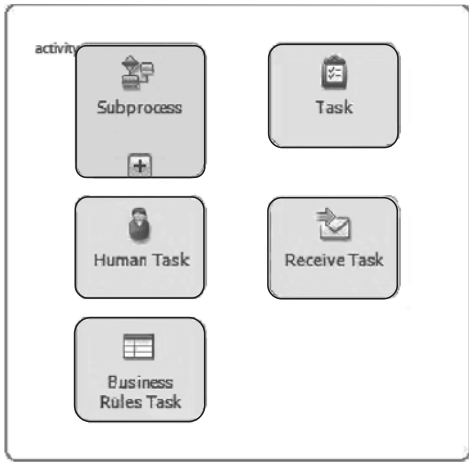

Рис. 2. Деятельности

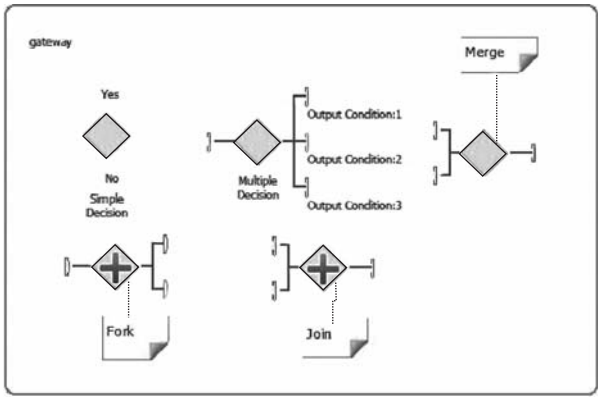

Рис. 3. Порты

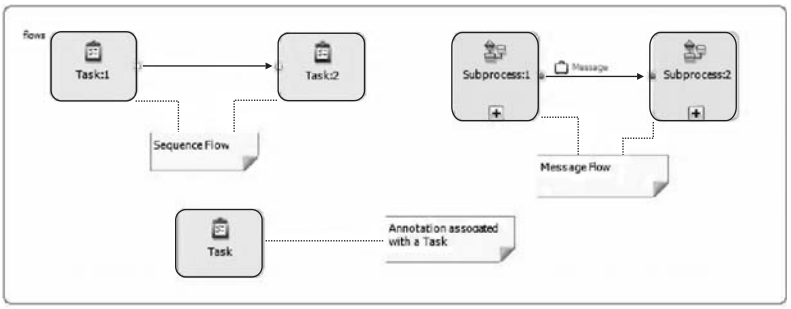

Рис. 4. Связывающие объекты

Объекты потока связаны между собой формированием скелета бизнес-процесса. Имеются три вида связки (рис. 4).

• Sequence Flow (поток последовательности) указывает порядок исполнения деятельности процесса. В BPMN не используется более общий термин «поток управления».

• Message Flow (поток сообщения) указывает обмен сообщениями между двумя процессами-участниками (Process Participants). Участниками могут быть бизнес-сущности или бизнес-роли. В BPMN пул (Pool) представляет одного участника.

• Association (ассоциация) соединяет данные, текст и другие артефакты с объектами потока.

Две диаграммы, простой бизнес-процесс и фрагмент более сложного бизнес-процесса, представлены на рис. 5 и 6.

Некоторые методологии моделирования бизнес-процессов используют концепцию коридоров как механизма организации деятельностей в виде визуально разделенных категорий для иллюстрации различных функциональных способностей и ответственностей. В BPMN поддерживаются два вида конструкций коридоров.

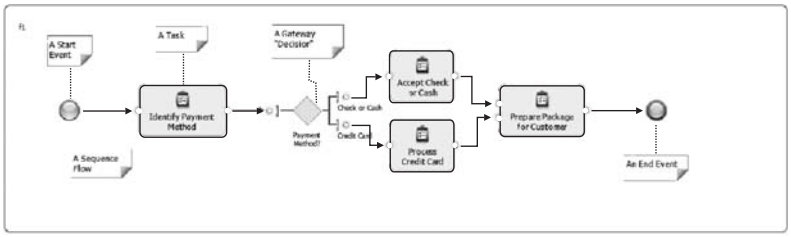

Рис. 5. Простой бизнес-процесс

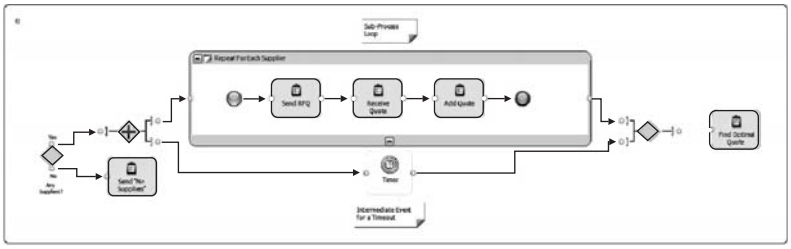

Рис. 6. Фрагмент сложного бизнес-процесса

• Pool (пул) представляет участников процесса. Он является контейнером для разделения множеств деятельности друг от друга. Обычно применяется в контексте B2B.

• Lane (лента) разделяет пул и располагается горизонтально или вертикально внутри него. Ленты применяются для организации или категоризации деятельностей (рис. 7).

Деятельности, организованные в отдельные пулы, можно рассматривать как самосодержащиеся процессы. Поток последовательности не должен пересекать границы пула. Поток сообщения является механизмом коммуникации между участниками, он связывает пулы или объекты в них.

Ленты более связаны с традиционным моделированием коридоров. Они применяются для обособления деятельностей данной функции или их роли. Поток последовательностей может пересекать границы ленты внутри одного пула. Не допускается поток сообщения между лентами одного и того же пула.

В диаграмму можно добавлять произвольное количество артефактов — столько, сколько нужно для моделирования бизнес-процесса. Пока поддерживаются три вида артефактов (рис. 8).

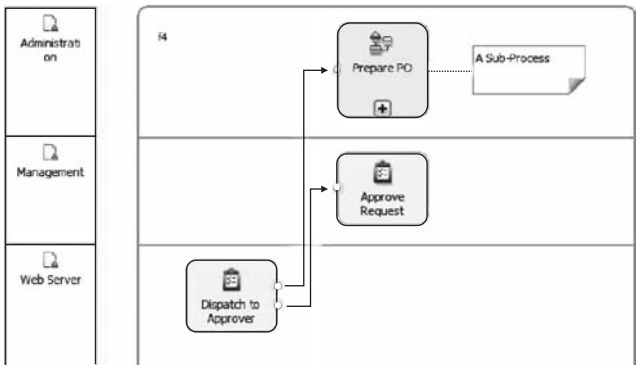

Рис. 7. Ленты

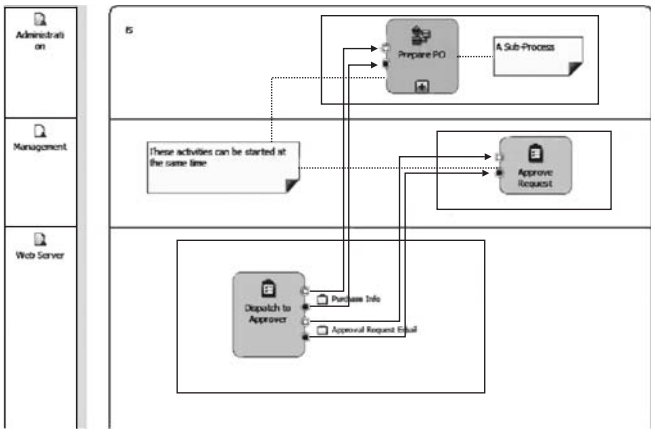

Рис. 8. Артефакты

• Data Object (объект данных) указывает на то, что необходимы данные или что производятся данные. Они связываются с деятельностями при помощи ассоциации.

• Group (группа) применяется для документирования или анализа, но не влияет на поток последовательности.

• Annotation (аннотация) дает добавочную текстовую информацию в диаграмме в помощь ее читателям.

Артефакты не изменяют основную структуру процесса.

В основном при помощи BPMN создаются два типа модели:

• сотрудничество (публичные) B2B-процессы,

• внутренние (частные) бизнес-процессы.

Сотрудничество предполагает взаимодействие нескольких процессов. Диаграммы этих процессов разрабатываются с глобальной точки зрения, они не принимают сторону одного участника, а показывают взаимодействие между всеми участниками. Взаимодействия представляются как последовательность деятельностей и образцов обмена сообщениями между участниками. Деятельности участников сотрудничества являются и «точками контакта» между ними. Процесс определяет все публично видимые деятельности всех участников. Процесс, представленный в одном пуле (один участник), в каком-то смысле абстрактный процесс, так как настоящий (внутренний) процесс обычно содержит больше деятельностей и деталей, чем видно на диаграммах рассматриваемого типа. На рис. 8 представлен пример сотрудничества.

Внутренние бизнес-процессы разрабатываются с точки зрения отдельной бизнес-организации. Несмотря на то, что эти процессы содержат взаимодействия с внешними участниками, они, в общем, содержат и много дея-

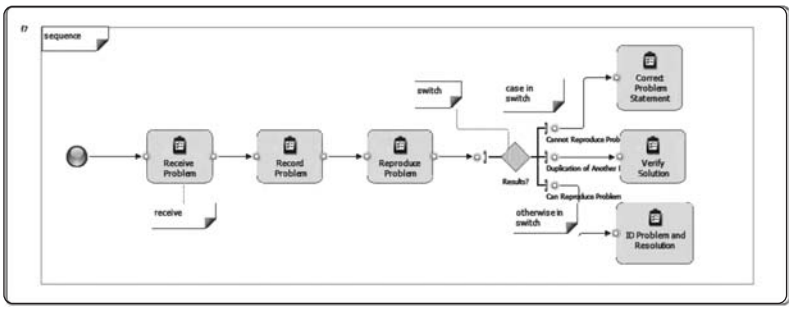

Рис. 9. Преобразование BPMN в WS-BPEL

тельностей, которые публично недоступны, т. е. частные деятельности. Если применяются ленты, то это могут быть только ленты одного пула. Поток последовательности не выходит за границы пула. Поток сообщений может пересекать границы пула для взаимодействия с другими внутренними бизнес-процессами. В одной диаграмме может быть представлено несколько частных бизнес-процессов.

Постепенная детализация в BPNM поддерживается вложением процессов. На рис. 9 показана идея преобразования спецификаций BPMN в процессы WS-BPEL.

ПРИЛОЖЕНИЕ 2. WS-BPEL

Язык WS-BPEL предназначен для описания поведенческих интерфейсов услуг и исполнения процессов, реализованных на сервисах. Поведенческий интерфейс (абстрактный процесс) является поведенческой спецификацией класса сервисов, которая включает ограничения порядка обмена сообщений между сервисами. Исполнимый процесс определяет порядок исполнения множества деятельностей (в основном коммуникационных); партнеров, участвующих в процессе; сообщения обмена; обработку событий и исключения.

Определение WS-BPEL-процесса состоит из множества деятельностей. Они подразделяются на две категории: основные и структурирующие. Основные деятельности атомарные:

- invoke вызывает операцию веб-сервиса;
- receive ожидает сообщения от внешнего партнера;
- reply отвечает внешнему партнеру;
- assign присоединяет стоимость переменной;
- throw сигнализирует об ошибке исполнения;
- compensate отменяет результат законченных деятельностей;

• exit прекращает исполнение экземпляра услуги;

• empty ничего не делает.

Структурирующие деятельности задают ограничения на поведение исполнения, содержащихся в них деятельностей:

• sequence задает последовательное исполнение;

• switch переключает ход исполнения;

• pick управляет состязанием на получение события и события таймера;

• while организует цикл;

• scope группирует деятельности в блок с единой обработкой исключений, авариями и компенсациями.

Структурирующие деятельности могут быть вставлены друг в друга или скомбинированы произвольным способом. Это позволяет строить сложные WS-BPEL-процессы.

Конструкции sequence, pick и while используются для выражения зависимостей структурированного потока управления. Кроме них в WS-BPEL имеется конструкция контрольных связок, которая вместе с нотацией условия соединения и условия перехода определяют порядок, синхронизацию и условные зависимости деятельностей, охваченных этой конструкцией. Наличие контрольной связки из деятельности «А» к деятельности «Б» означает, что «Б» может начать свое исполнение только после окончания исполнения «А» или после ее отклонения. Кроме того, «Б» может быть исполнено, только если истинно его условие соединения, иначе «Б» отклоняется. Условие соединения задается в терминах маркеров, переносимых на контрольные связки — те, которые ведут к «Б». Эти маркеры могут иметь положительную стоимость (истина) или отрицательную стоимость (ложь). Деятельность «Х» генерирует положительный маркер в выходящую из него связку «С»:

• если «Х» исполнена (не отклонена) и

• условие перехода, ассоциированное из связки «С», исчисляется в истину.

Условия перехода — это логические выражения на переменных процесса (как и условия деятельности switch). Процесс (обработка), который распространяет положительные и отрицательные маркеры в контрольные связки и который определяет исполнение или отклонение деятельностей, называется устранением тупиков. Контрольные связки могут пересекать границы почти всех структурирующих деятельностей. При этом они не должны создавать циклические контрольные зависимости и не должны пересекать границ деятельности while или сериализуемого scope. Взаимодействие между структурирующими деятельностями и контрольными связками не имеет однозначного определения у пользователей, в результате этого возникают противоречивые WS-BPEL-спецификации.

Несмотря на то, что конструкции потока управления WS-BPEL разрабатывались для исполнения WS-BPEL-процесса без входа в тупик, некоторые структурирующие конструкции (например, switch и pick) совместно с контрольными связками могут привести к тому, что некоторые деятельности будут недостижимы.

Другая группа конструкций потока управления в WS-BPEL включает обработчиков событий, аварий и компенсаций. Обработчик события — это правило «событие–действие» в определенном обхвате. Обработчик события активен, когда его обхват находится в состоянии исполнения, и он может исполняться конкурентно с основной деятельностью обхвата. Когда появляется событие, ассоциированное с обработчиком (это может быть событие таймера или получение сообщения), начинается исполнение тела обработчика и продолжается исполнение основной деятельности обхвата. Обработчики аварии задают реакцию на внешние или внутренние аварии во время исполнения обхвата. Некоторые аварии могут быть выброшенными деятельностью throw. Обработчики аварии не исполняются в конкуренции с основной деятельностью обхвата. Вместо этого основная деятельность прекращается до начала исполнения обработчика аварии. Наконец, обработчики компенсации, которые связаны с деятельностью compensate, позволяют, чтобы процесс отменил работу уже закончившегося обхвата. Когда исполняется деятельность в некотором обхвате, она активирует обработчика компенсации этого обхвата, если такой есть в наличии. Это может включать исполнение обработчиков компенсации, ассоциированных только под рассматриваемый обхват.

Пример. WSDL-определения, использованные в примере (рис. 10, 11):

```
<wsdl:definitions
 targetNamespace=
  "http://manufacturing.org/wsdl/purchase"
xmlns:sns="http://manufacturing.org/xsd/purchase"
xmlns:pos="http://manufacturing.org/wsdl/purchase"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:plnk=
 "http://docs.oasis-open.org/wsbpel/2.0/plnktype"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
 <xsd:schema>
  <xsd:import
   namespace="http://manufacturing.org/xsd/purchase"
   schemaLocation=
    "http://manufacturing.org/xsd/purchase.xsd" />
  </xsd:schema>
</wsdl:types>
<wsdl:message name="POMessage">
 <wsdl:part name="customerInfo"
  type="sns:customerInfoType" />
 <wsdl:part name="purchaseOrder"
  type="sns:purchaseOrderType" />
```

```
</wsdl:message>
<wsdl:message name="InvMessage">
 <wsdl:part name="IVC" type="sns:InvoiceType" />
</wsdl:message>
<wsdl:message name="orderFaultType">
 <wsdl:part name="problemInfo"
  element="sns:OrderFault" />
</wsdl:message>
<wsdl:message name="shippingRequestMessage">
 <wsdl:part name="customerInfo"
  element="sns:customerInfo" />
</wsdl:message>
<wsdl:message name="shippingInfoMessage">
 <wsdl:part name="shippingInfo"
  element="sns:shippingInfo" />
</wsdl:message>
<wsdl:message name="scheduleMessage">
 <wsdl:part name="schedule"
  element="sns:scheduleInfo" />
</wsdl:message>
<!-- portTypes supported by the purchase order
 process -->
 <wsdl:portType name="purchaseOrderPT">
  <wsdl:operation name="sendPurchaseOrder">
  <wsdl:input message="pos:POMessage" />
  <wsdl:output message="pos:InvMessage" />
  <wsdl:fault name="cannotCompleteOrder"
   message="pos:orderFaultType" />
 </wsdl:operation>
</wsdl:portType>
 <wsdl:portType name="invoiceCallbackPT">
  <wsdl:operation name="sendInvoice">
 <wsdl:input message="pos:InvMessage" />
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="shippingCallbackPT">
 <wsdl:operation name="sendSchedule">
  <wsdl:input message="pos:scheduleMessage" />
 </wsdl:operation>
</wsdl:portType>
<!-- portType supported by the invoice services -->
<wsdl:portType name="computePricePT">
 <wsdl:operation name="initiatePriceCalculation">
  <wsdl:input message="pos:POMessage" />
 </wsdl:operation>
 <wsdl:operation name="sendShippingPrice">
  <wsdl:input message="pos:shippingInfoMessage" />
 </wsdl:operation>
</wsdl:portType>
<!-- portType supported by the shipping service -->
<wsdl:portType name="shippingPT">
 <wsdl:operation name="requestShipping">
```

```
  <wsdl:input message="pos:shippingRequestMessage" />
  <wsdl:output message="pos:shippingInfoMessage" />
  <wsdl:fault name="cannotCompleteOrder"
   message="pos:orderFaultType" />
 </wsdl:operation>
</wsdl:portType>
<!-- portType supported by the production
 scheduling process -->
<wsdl:portType name="schedulingPT">
 <wsdl:operation name="requestProductionScheduling">
  <wsdl:input message="pos:POMessage" />
 </wsdl:operation>
 <wsdl:operation name="sendShippingSchedule">
   <wsdl:input message="pos:scheduleMessage" />
  </wsdl:operation>
 </wsdl:portType>
 <plnk:partnerLinkType name="purchasingLT">
  <plnk:role name="purchaseService"
   portType="pos:purchaseOrderPT" />
 </plnk:partnerLinkType>
 <plnk:partnerLinkType name="invoicingLT">
  <plnk:role name="invoiceService"
   portType="pos:computePricePT" />
  <plnk:role name="invoiceRequester"
   portType="pos:invoiceCallbackPT" />
 </plnk:partnerLinkType>
 <plnk:partnerLinkType name="shippingLT">
  <plnk:role name="shippingService"
   portType="pos:shippingPT" />
  <plnk:role name="shippingRequester"
   portType="pos:shippingCallbackPT" />
 </plnk:partnerLinkType>
 <plnk:partnerLinkType name="schedulingLT">
  <plnk:role name="schedulingService"
   portType="pos:schedulingPT" />
 </plnk:partnerLinkType>
</wsdl:definitions>
```

WS-BPEL-код:

```
<process name="purchaseOrderProcess"
 targetNamespace="http://example.com/ws-bp/purchase"
 xmlns="http://docs.oasis-open.org/wsbpel/2.0/
 process/executable"
xmlns:lns="http://manufacturing.org/wsdl/purchase">
<documentation xml:lang="EN"$>$

   A simple example of a WS-BPEL process for handling a purchase
order.

</documentation>
<partnerLinks>
 <partnerLink name="purchasing"
```

```
  partnerLinkType="lns:purchasingLT"
  myRole="purchaseService" />
 <partnerLink name="invoicing"
  partnerLinkType="lns:invoicingLT"
  myRole="invoiceRequester"
  partnerRole="invoiceService" />
 <partnerLink name="shipping"
  partnerLinkType="lns:shippingLT"
  myRole="shippingRequester"
  partnerRole="shippingService" />
 <partnerLink name="scheduling"
  partnerLinkType="lns:schedulingLT"
  partnerRole="schedulingService" />
</partnerLinks>
<variables>
 <variable name="PO" messageType="lns:POMessage" />
 <variable name="Invoice"
  messageType="lns:InvMessage" />
 <variable name="shippingRequest"
  messageType="lns:shippingRequestMessage" />
 <variable name="shippingInfo"
  messageType="lns:shippingInfoMessage" />
 <variable name="shippingSchedule"
  messageType="lns:scheduleMessage" />
</variables>
<faultHandlers>
 <catch faultName="lns:cannotCompleteOrder"
  faultVariable="POFault"
  faultMessageType="lns:orderFaultType">
  <reply partnerLink="purchasing"
   portType="lns:purchaseOrderPT"
   operation="sendPurchaseOrder"
   variable="POFault"
  faultName="cannotCompleteOrder" />
 </catch>
</faultHandlers>
<sequence>
 <receive partnerLink="purchasing"
  portType="lns:purchaseOrderPT"
  operation="sendPurchaseOrder"
  variable="PO"
  createInstance="yes">
   <documentation>

    Receive Purchase Order

   </documentation>
 </receive>
 <flow>
   <documentation>
```

```
   A parallel flow to handle shipping, invoicing and scheduling

  </documentation>
  <links>
   <link name="ship-to-invoice" />
   <link name="ship-to-scheduling" />
  </links>
  <sequence>
   <assign>
    <copy>
     <from>$PO.customerInfo</from>
     <to>$shippingRequest.customerInfo</to>
    </copy>
   </assign>
   <invoke partnerLink="shipping"
    portType="lns:shippingPT"
    operation="requestShipping"
    inputVariable="shippingRequest"
    outputVariable="shippingInfo">
    <documentation>


   Decide On Shipper


  </documentation>
  <sources>
   <source linkName="ship-to-invoice" />
  </sources>
 </invoke>
 <receive partnerLink="shipping"
  portType="lns:shippingCallbackPT"
  operation="sendSchedule"
  variable="shippingSchedule">
  <documentation>


   Arrange Logistics


  </documentation>
  <sources>
   <source linkName="ship-to-scheduling" />
  </sources>
 </receive>
</sequence>
<sequence>
 <invoke partnerLink="invoicing"
 portType="lns:computePricePT"
 operation="initiatePriceCalculation"
 inputVariable="PO">
 <documentation>
```

```
    Initial Price Calculation

   </documentation>
  </invoke>
  <invoke partnerLink="invoicing"
   portType="lns:computePricePT"
   operation="sendShippingPrice"
   inputVariable="shippingInfo">
   <documentation>

    Complete Price Calculation

   </documentation>
   <targets>
    <target linkName="ship-to-invoice" />
   </targets>
  </invoke>
  <receive partnerLink="invoicing"
   portType="lns:invoiceCallbackPT"
   operation="sendInvoice"
   variable="Invoice" />
 </sequence>
 <sequence>
  <invoke partnerLink="scheduling"
   portType="lns:schedulingPT"
   operation="requestProductionScheduling"
   inputVariable="PO">
   <documentation>

    Initiate Production Scheduling

   </documentation>
  </invoke>
  <invoke partnerLink="scheduling"
   portType="lns:schedulingPT"
   operation="sendShippingSchedule"
   inputVariable="shippingSchedule">
   <documentation>

    Complete Production Scheduling

     </documentation>
     <targets>
      <target linkName="ship-to-scheduling" />
     </targets>
    </invoke>
   </sequence>
  </flow>
  <reply partnerLink="purchasing"
   portType="lns:purchaseOrderPT"
   operation="sendPurchaseOrder"
   variable="Invoice">
```

```
   <documentation>Invoice Processing</documentation>
  </reply>
 </sequence>
</process>
```

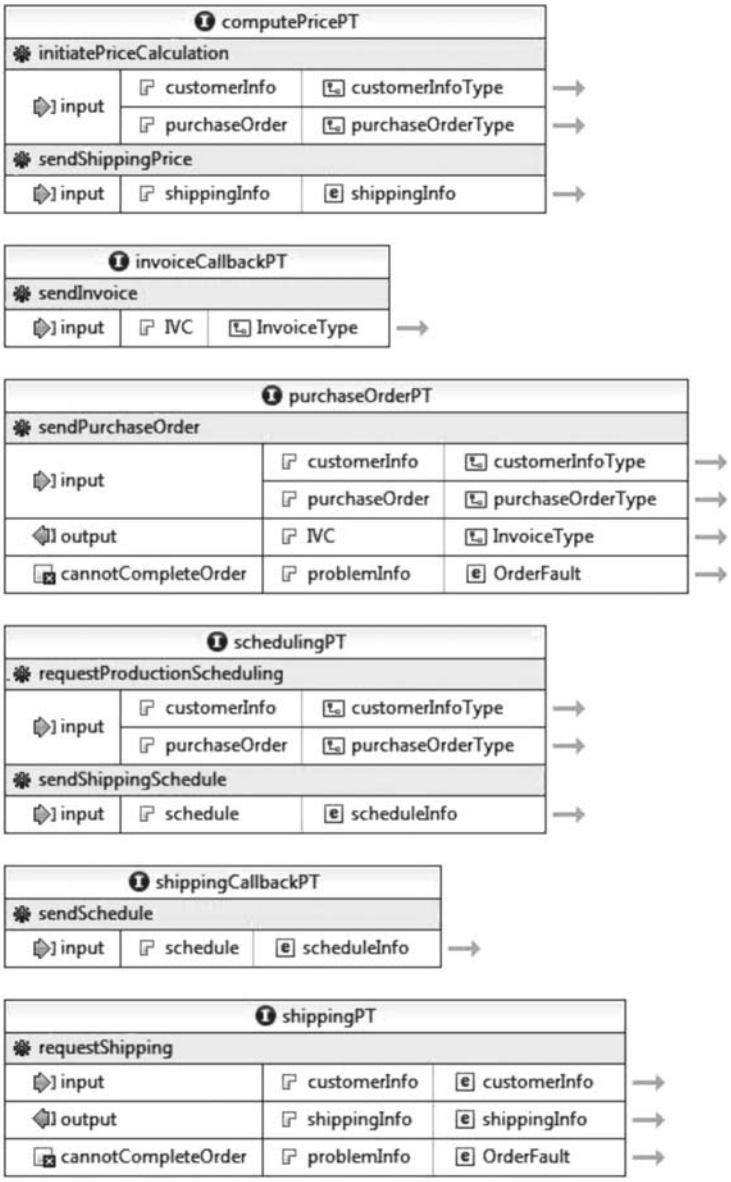

Рис. 10. Сервисы, использованные в примере, на диаграмме IBM Rational Software Architect

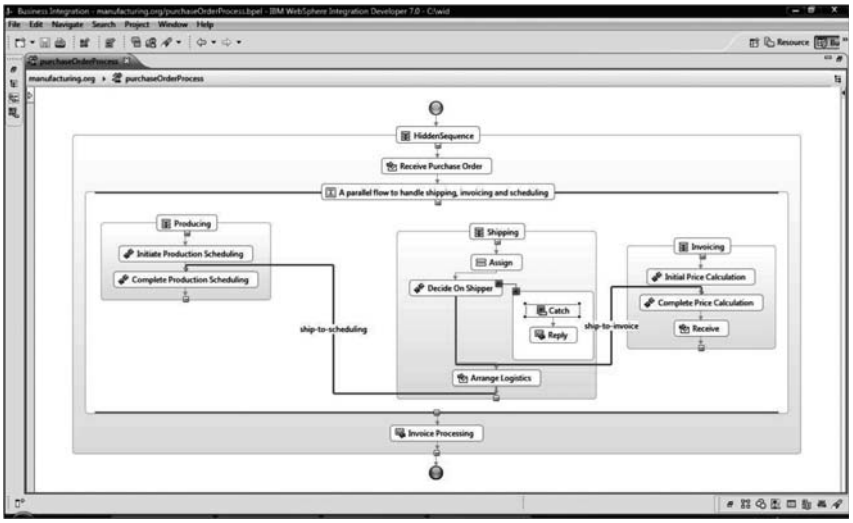

Рис. 11. WS-BPEL-код представлен в виде диаграммы IBM WebSphere Integration Developer

ПРИЛОЖЕНИЕ 3. UML-ДИАГРАММА ДЕЯТЕЛЬНОСТИ

Диаграммы деятельности UML обычно применяются для моделирования бизнес-процессов, логики отдельного варианта применения, логики сценария пользования или детализации логики бизнес-правила. Независимо от того, что диаграммы деятельности можно использовать для моделирования внутренней логики сложной операции, рекомендуется перепроектировать операцию, чтобы она стала довольно проста и не требовала бы диаграммы деятельности.

Диаграмма деятельности представляет последовательность исполнения деятельности. Она расписывает поток управления от некоторой начальной точки до некоторой конечной точки или точек исполнения. При этом она детализирует те места, где принимаются решения на основе событий, случающихся во время исполнения деятельности. Детализация может включать и параллелизм исполнения. На рис. 12 представлен пример диаграммы деятельности.

Далее рассмотрим элементы диаграммы деятельности.

Деятельность (рис. 13) является спецификацией параметризированной последовательности поведения. Она охватывает все действия, управление потоком исполнения и все другие элементы деятельности.

Действие — это отдельный шаг деятельности (рис. 14).

Ограничения деятельности связываются с ним. На примере представлены пре- и постусловия (рис. 15).

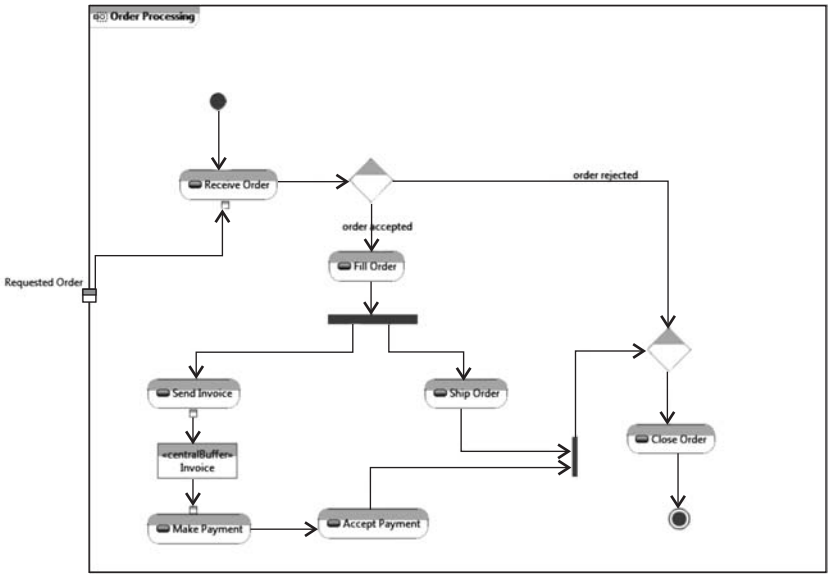

Рис. 12. Диаграмма деятельности исполнения заказа

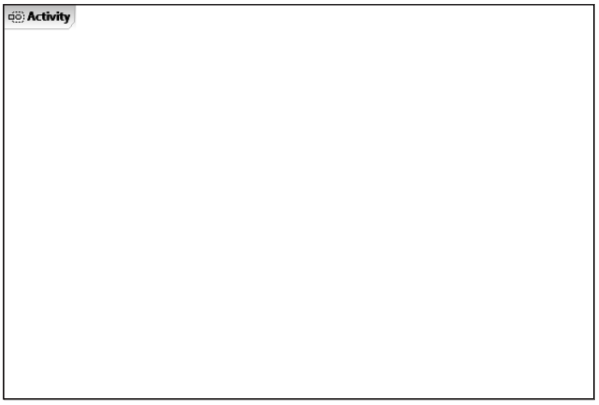

Рис. 13. Деятельность

Поток управления (рис. 16) указывает на то, что управление передается с одного действия на другое после завершения первого действия.

Начальный узел указывает на начало деятельности (рис. 17).

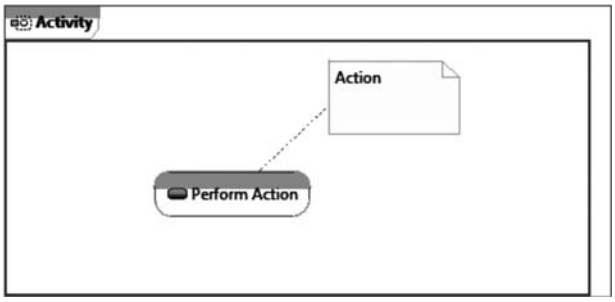

Рис. 14. Действие

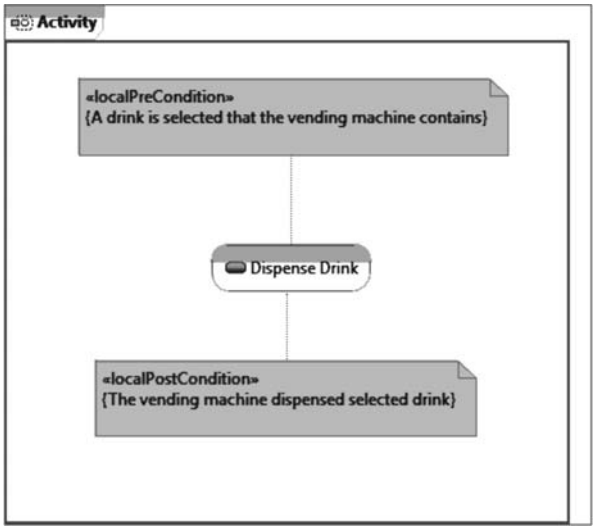

Рис. 15. Действие с ограничениями

Имеются два вида конечных узлов: конец деятельности и конец потока (рис. 18). Конец потока указывает на конец отдельного потока управления. Конец деятельности указывает на конец всех потоков управления деятельности.

Поток объектов является путем, по которому передаются или получаются объекты (рис. 19). Поток объектов указывает направление передачи объектов. По крайней мере в одном конце объектного потока должен находиться объект. Кроме того, объектный поток можно представить в сокращенном виде с применением входных и выходных точек.

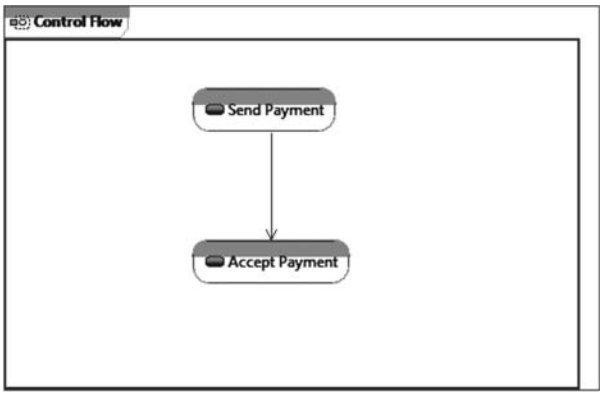

Рис. 16. Поток управления

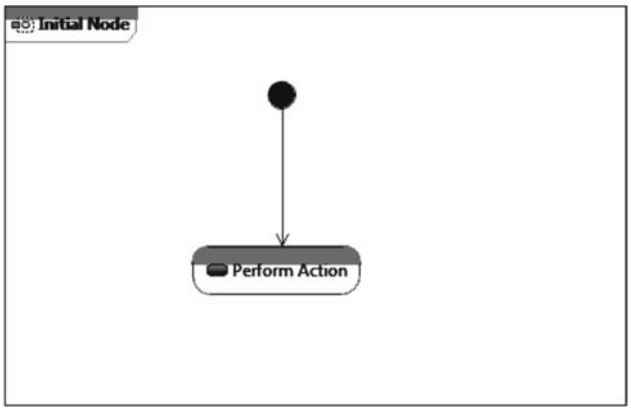

Рис. 17. Начальный узел

Склад данных — это объект со стереотипом (рис. 20).

Узлы принятия решения и узлы слияния (рис. 21) имеют одну и ту же нотацию — ромб. Они могут быть именованы. Поток управления, который выходит из узла решения, должен иметь условие защиты, которое определяет, когда поток управления имеет право проходить через это разветвление.

Развилки и соединения (рис. 22) также имеют одну и ту же нотацию: вертикальный или горизонтальный блок — это связано с направлением развития диаграммы: слева направо или сверху вниз. Они указывают на начало и конец конкурентных нитей управления.

Отличие соединения от слияния в том, что оно синхронизирует все входящие потоки, прежде чем дать выход. При слиянии входной поток управления

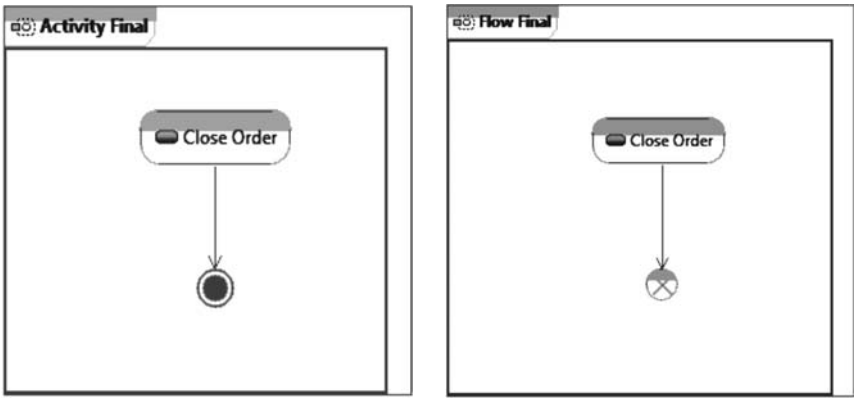

Рис. 18. Конец деятельности и конец потока

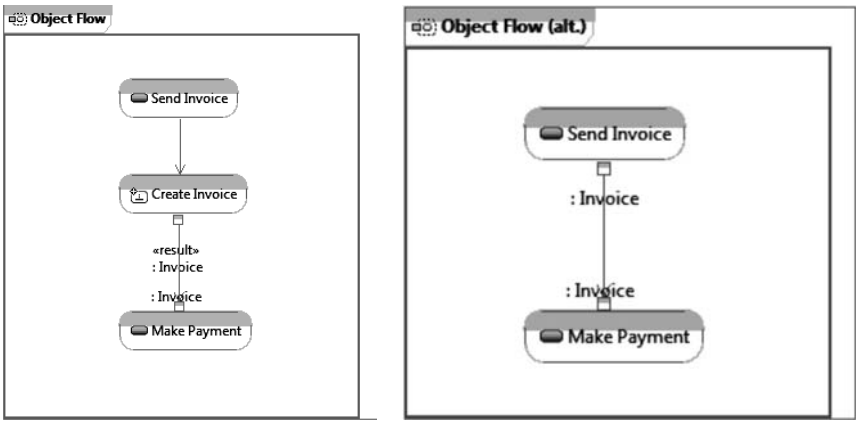

Рис. 19. Поток объектов

Рис. 20. Склад данных

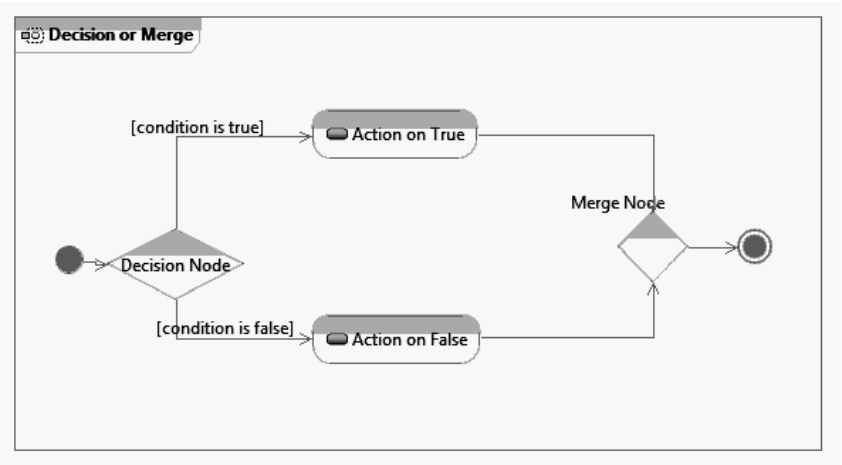

Рис. 21. Решение и слияние

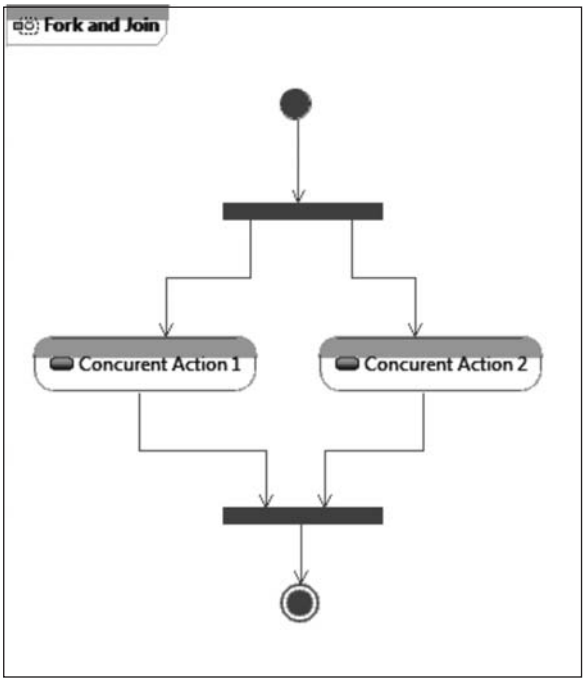

Рис. 22. Развилка и соединение

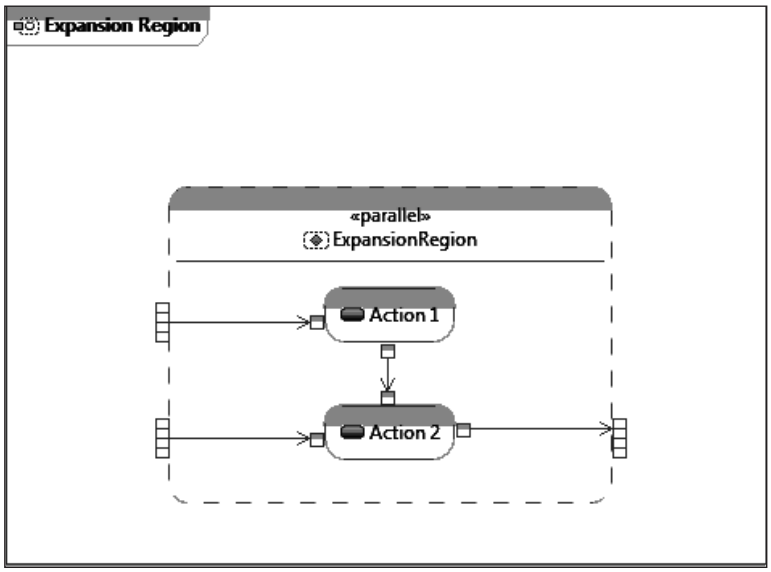

Рис. 23. Регион расширения

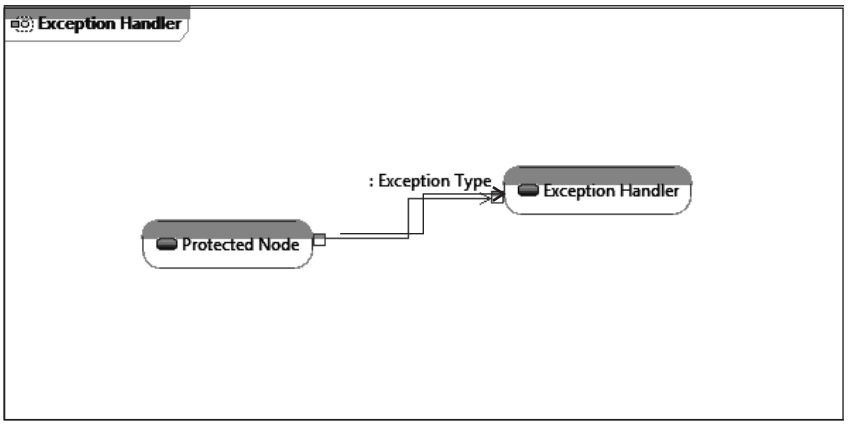

Рис. 24. Обработчик исключений

не задерживается и свободно проходит. Если одновременно существует несколько входных потоков, то действие, которое находится на выходе слияния, исполнится столько раз, сколько есть входящих потоков.

Регион расширения — это структурированная деятельность, которая исполняется много раз (рис. 23). Вид региона указан ключевыми словами: iterative, parallel или stream.

Обработчик исключения можно моделировать, как показано на рис. 24.

Регион прерывания содержит группу деятельностей, которая может быть прервана (рис. 25).

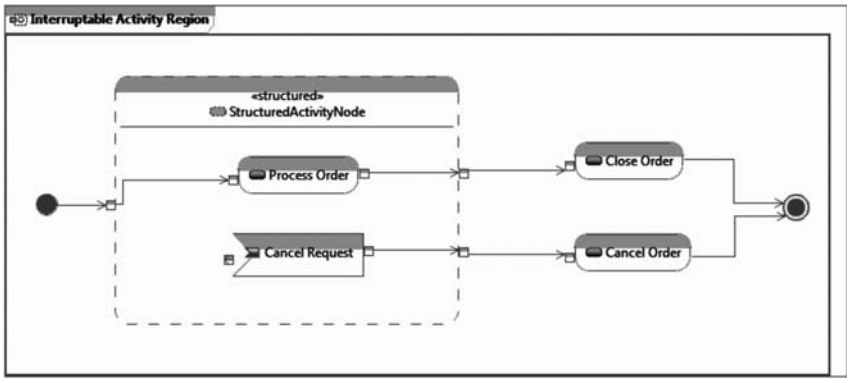

Рис. 25. Регион прерывания

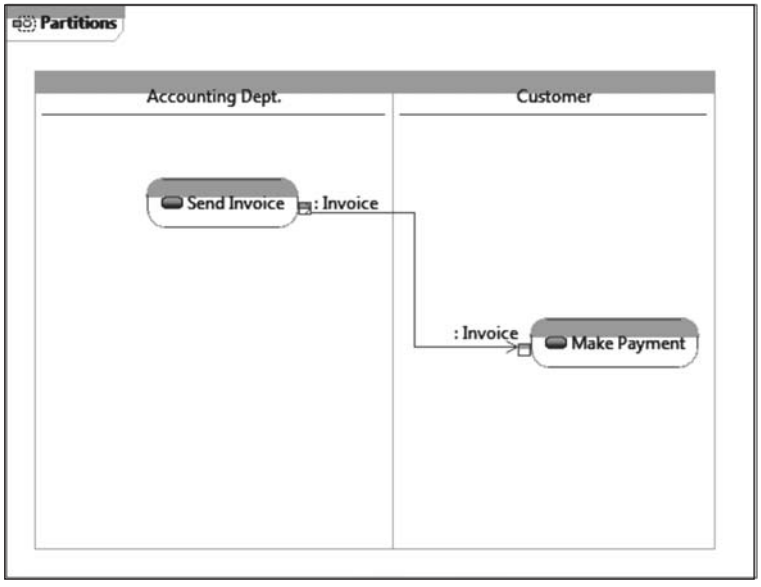

Рис. 26. Коридоры

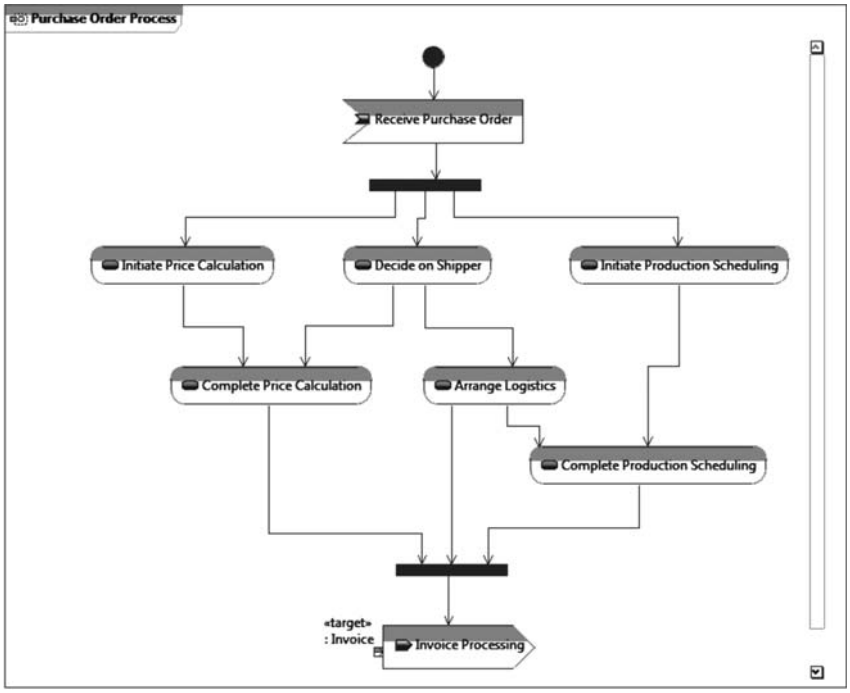

Рис. 27. Бизнес-процесс

Раздел действия можно представить в виде горизонтальных или вертикальных коридоров, обычно это делается по организационным единицам (рис. 26).

Пример полного бизнес-процесса представлен на рис. 27.

ПРИЛОЖЕНИЕ 4. WS-CDL

С концептуальной точки зрения структура WS-CDL показана на рис. 28.

Хореография описания WS-CDL обеспечивает глобальное или беспристрастное представление взаимодействий между несколькими участниками. Веб-сервисы разрабатываются разными поставщиками, и требуется описание способов их использования совместно с другими сервисами, которое невозможно достичь, используя WSDL (Web Service Description Language). Рабочая группа W3C Choreography была создана из-за необходимости получения средств описания совместного использования веб-сервисов для достижения определенных целей.

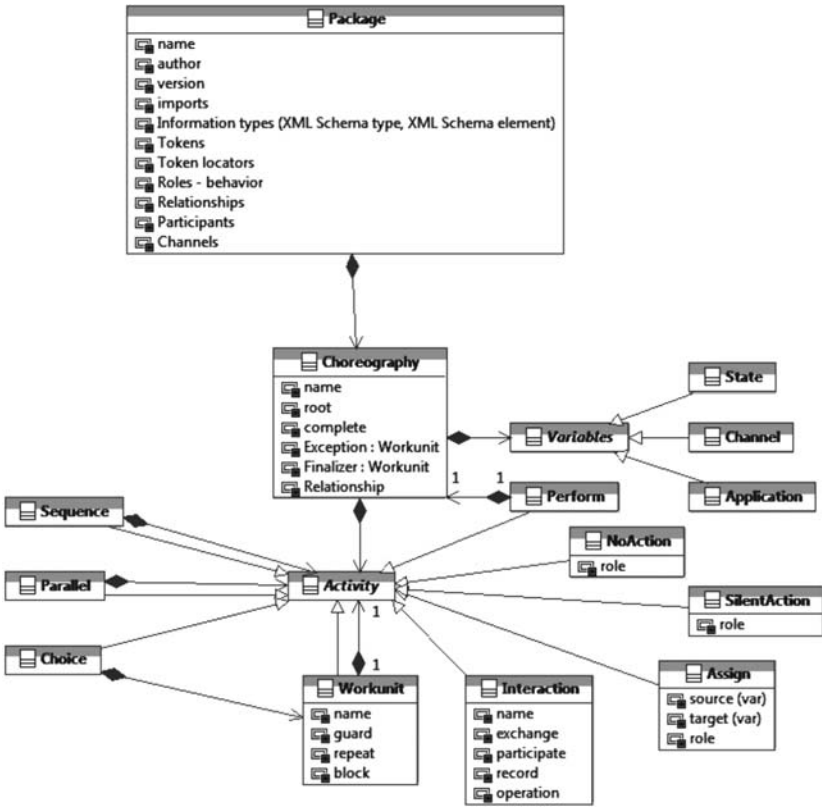

Рис. 28. WS-CDL-пакет

Назначение WS-CDL-спецификации — описывать контракты между участниками, представляя внешне наблюдаемое поведение веб-сервисов и их клиентов (обычно другие веб-сервисы), и описывать обмен сообщениями между ними. По замыслу должна быть возможность из описания WS-CDL-хореографии генерировать скелет кода веб-сервисов или абстрактных процессов WS-BPEL.

Описание хореографии представляет пакет, являющийся контейнером для набора деятельностей, исполняемых участниками. Основные типы деятельностей: деятельности потока управления (control flow), деятельности WorkUnit и основные деятельности (basic activities).

Первая категория деятельностей включает Sequence, Parallel и Choice. Деятельность Choice выбирает исполнение некоторой деятельности из фик-

сированного множества конкурентных деятельностей. Эта деятельность осуществляет два вида выбора.

• Выбор, управляемый данными (непосредственный выбор). В этом случае выбор осуществляется исчислением логического условия переменных данных. Исчисление осуществляется, когда деятельность Choice является управляющей.

• Выбор, управляемый событиями (отложенный выбор). Выбор в этом случае определяется появлением некоторого события среди фиксированного множества конкурентных событий. Событие может быть генерировано в результате изменения переменных из-за взаимодействия или исполнения некоторой деятельности в хореографии. Изменение переменных приводит к новому результату исчисления некоторых условий. Выбор отложен, когда управление передается деятельности выбора, исполнение приостанавливается в ожидании события.

Тип деятельности выбора определяется природой деятельностей множеств выбора. Если все деятельности имеют защитные неблокирующие условия, то выбор управляется данными. Иначе выбор может быть полностью управляемым событиями или комбинацией обоих типов. В этом WS-CDL отличается от WS-BPEL, где имеются только два типа выбора в отдельных конструкциях (switch и pick).

Вторым типом деятельности в WS-CDL является WorkUnit. Эта деятельность описывает условное или, возможно, циклическое исполнение некоторой деятельности. Ее синтаксис состоит из нескольких частей, включающих референцию к исполняемой деятельности, защитное условие блока и условие цикла — это условия логического типа, которые определяют, сколько раз будет исполняться деятельность (возможно, ни разу). Исполнение деятельности WorkUnit начинается исчислением защитного условия. Если получается «истина», то деятельность исполняется, в другом случае — пропускается. После исполнения деятельности исчисляется условие цикла и в зависимости от полученного результата происходит исполнение деятельности. Вложением друг в друга деятельностей WorkUnit можно организовать вложение циклов. Условие блока используется для задержки исчисления защитного условия и условия цикла до тех пор, пока доступны все используемые ими переменные. Обычно WorkUnit используется совместно с деятельностью Choice, и в этом случае выбор может быть непосредственным или отложенным в зависимости от условия блока WorkUnit, охваченного деятельностью выбора.

Деятельности Sequence, Parallel, Choice и WorkUnit являются типичными конструкциями процедурного языка программирования. Эти деятельности соответствуют деятельностям Sequence, Flow, While, Switch и Pick языка WS-BPEL. Только изображение деятельностей Choice и WorkUnit нетривиально, так как существует необходимость выбора между Switch и Pick с учетом блока условия.

Третий тип WS-CDL-деятельностей — это основные деятельности: Interaction, NoAction, SilentAction, Assign и Perform. Деятельности NoAction и SilentAction задают те места в хореографии, где некоторая именованная роль исполняет или не исполняет «за кулисами» действие, которое не влияет на хореографии. Деятельность Assign используется для передачи стоимости одной переменной другой, когда обе переменные находятся в одной и той же роли. Деятельность Perform используется для вызова другой хореографии к исполнению в контексте текущей хореографии. Вызываемая хореография может начинаться с того же самого пакета, с которого и вызывающая, или может быть из совсем другого пакета, импортированного в текущий пакет. Деятельность исполнения поддерживает механизм увязки переменными вызывающей хореографии свободными переменными вызываемой хореографии.

Самым важным элементом WS-CDL является деятельность Interaction. Эта деятельность описывает обмен информацией между участниками с выделением получателя. Взаимодействия могут быть следующими: запрос (request), ответ (respond) или запрос-ответ (request-respond). Описание взаимодействия состоит из трех частей: участники, обмениваемая информация и канал обмена информацией. Ниже приводится пример взаимодействия. Надо отметить асимметричную природу описания — внимание уделяется получателю. Точнее, WS-CDL описывает, какие операции предпринимаются при получении информации, а не то, что делается до ее посылки.

```
   <?xml version="1.0" encoding="utf-8"?>
   <interaction name="QuoteResponse" initiate="false" align=
"false"
          operation="ReceiveQuote" channelVariable="receiveQuote
Channel">
    <participate toRole="customer" fromRole="supplier"
          relationship="GetQuote" />
    <exchange name="e2" action="respond"
         informationType="quoteDocument">
     <send variable="quote" />
     <receive variable="quote" />
    </exchange>
   </interaction>
```

Роли from (посылающий), to (получающий) и отношение между ними явным образом задаются в разделе participate. Несмотря на то, что отношение задано ролями, для некоторого поведения возможно, что операция, которая будет исполняться при приеме информации, задана на другом месте определения взаимодействия, а не в разделе participate.

Информация взаимодействия, которая посылается и получается, задается переменными. Каждая переменная описывается своими элементами informationType и recordReference. Три вида информации представлены переменными: информация, зависимая от приложения; информация о состоянии; ин-

формация о канале. Содержание переменных доступно функциями WS-CDL расширением XPath 1.0, например, getVariable(name, path, role).

Существуют также переменные канального типа. Этот тип задает роль получателю (сообщения типа заказ или ответ) и поведение получателя. Поведение роли можно задавать также WSDL-интерфейсами.

```
<channelType name= "requestQuoteChannel" action= "request"
      usage= "unlimited">
      <role type= "supplier" behavior= "ReceiveRFQ"/>
      <reference>
             <token name= "supplierRef"/>
      </reference>
      <identity>
              <token name= "quoteId"/>
      </identity>
      <passing action= "respond" new= "true"
             channel= "receiveQuoteChannel"/>
</channelType>
<channelType name= "receiveQuoteChannel" action= "respond"
      usage= "once">
      <role type= "customer" behavior= "ReceiveQuote"/>
      <reference>
             <token name= "customerRef"/>
      </reference>
      <identity>
             <token name= "quoteId"/>
      </identity>
</channelType>
```

Каналы не подразделяются на двусторонние и односторонние. Каждому каналу соответствует вид: заказ, ответ или заказ-ответ. В принципе, отношения между этими тремя видами каналов и WSDL Message Exchange Patterns нуждаются в разработке.

Во всяком случае, каналы — это связь между WS-CDL-хореографиями и WSDL-интерфейсами. Каждое поведение в хореографии описывается канальным типом. Переменные канального типа имеют только одно поведение. По каналам передаются переменные состояния, переменные типа приложения и переменные канального типа. В описании канального типа указываются имена канальных типов, которые можно передавать в канале рассматриваемого типа.

Маркер reference указывает на поиск получателя, а маркер identity используется для корреляции обмениваемых сообщений. Маркер представлен как информационный тип. Локаторы маркеров дают возможность находить маркеры с применением XPath-выражений.

Маркеры и информационные типы задаются на уровне пакета и доступны всем хореографиям и деятельностям пакета. Информационные типы могут быть XML Schema элементы, WSDL 2.0 элементы схемы, XML Schema типы

или WSDL 1.1 типы сообщения. Все виды переменных имеют какой-то информационный тип. Переменные, используемые деятельностями хореографий, определяются на уровне всей хореографии. Эти переменные доступны для совместного использования с вызванными хореографиями с применением механизма увязки деятельности perform.

Переменные могут одновременно принадлежать нескольким ролям хореографии. Передачу стоимости переменных между ролями можно делать сообщениями деятельности взаимодействия.

```
<informationType name= "address" type= "xsd:anyURI"/>
<informationType name= "correlationId" type= "xsd:string"/>
<informationType name= "quoteRequest" type= "xsd:anyURI"/>
<informationType name= "quoteDocument" type= "xsd:anyURI"/>
<token name= "customerRef" informationType= "address"/>
<token name= "supplierRef" informationType= "address"/>
<token name= "quoteId" informationType= "correlationId"/>
<tokenLocator tokenName= "quoteId" query= "quoteRequest/docId"
      informationType= "quoteRequest">
</tokenLocator>
```

Описание хореографии содержит деятельности самого высокого уровня. Этот контейнер может иметь обработчик исключения и компенсатор. Обработчик исключения активируется, когда возникает исключение. Компенсатор активируется, когда рассматриваемая хореография закончила свою работу в контексте некоторой другой хореографии, и в последней возникла ошибка — тогда компенсатор отменяет результаты рассматриваемой хореографии.

```
<choreography name= "QuoteAndOrder" isolation= "dirty-write"
   root= "true">
   <relationship type= "GetQuote"/>
   <relationship type= "PlaceOrder"/>
   <variableDefinitions>
          <variable name= "rfq" mutable= "false" free= "false"
                informationType= "quoteRequest"
                silentAction= "false"/>
          <variable name= "quoteForm" mutable= "true"
                free= "false" informationType= "quoteDocument"
                silentAction= "true" role= "supplier"/>
          <variable name= "quote" mutable= "true" free= "false"
                informationType= "quoteDocument"
                silentAction= "false"/>
          </variableDefinitions>
             <sequence> ... activities ... </sequence>
             <exception name= "NoAction">
                   <workunit block= "false" repeat= "false"
                        name= "NoAction" guard= "true">
                        <noAction role= "customer"/>
                   </workunit>
             </exception>
```

```
            <finalizer name= "None">
                    <workunit block= "false" repeat= "false"
name= "None"
                            guard= "true">
                            <noAction role= "customer"/>
                    </workunit>
            </finalizer>
   </choreography>
```

Пакет содержит несколько хореографий и используется для объединения общей информации обо всех содержащихся в нем хореографиях. Пакет описывает типы ролей, которые обладают поведением и типами отношения между двумя типами роли. Типы отношения задают подмножество ролей поведения, которое проявляется данным отношением. В конечном итоге пакет может содержать описание типов участников, которое осуществляет логическую группировку ролей. Каждая роль выявляет по крайней мере одно поведение. Отношение задает две роли и подмножество поведения для каждой роли среди подмножеств, использующихся в этой роли. Каждый участник может играть определенное множество ролей с определенными подмножествами поведения для каждой из этих ролей. Типы участников определяются на уровне пакета, но не используются в хореографиях и деятельностях. WS-CDL не рассматривает развертку статичного проекта (роли, поведения, отношения и участники) во время исполнения для поставщиков сервисов и конкретных сервисов.

```
   <package name= "Buying"
         xmlns= "http://www.w3.org/2004/10/ws-chor/cdl/"
   ...
         <roleType name= "supplier">
                <behavior interface= "Supplier.wsdl" name=
"ReceiveRFQ"/>
                <behavior interface= "Supplier.wsdl" name=
"ReceivePO"/>
         </roleType>
         <roleType name= "customer">
                <behavior interface= "Customer.wsdl"
                      name= "ReceiveQuote"/>
                <behavior interface= "Customer.wsdl"
                      name= "ReceivePaymentRequest"/>
         </roleType>
         <relationshipType name="GetQuote">
                <role type= "supplier" behavior= "ReceiveRFQ"/>
                <role type= "customer" behavior= "Receive
Quote"/>
         </relationshipType>
         <relationshipType name="PlaceOrder">
                <role type= "supplier" behavior= "ReceivePO"/>
                <role type= "customer" behavior= "ReceivePayment
```

```
Request"/>
         </relationshipType>
         <participantType name= "CustomerA">
                <role type= "customer"/>
         </participantType>
         <participantType name= "SupplierB">
                <role type= "supplier"/>
                <role type= "customer"/>
         </participantType>
   ...
   </package>
```

Несмотря на то, что все элементы WS-CDL имеют уникальные имена, многие из них (хореографии, деятельности, роли, отношения и каналы) нуждаются в уникальных идентификаторах экземпляров во время исполнения. Например, если в хореографии применяется взаимодействие с параллельной деятельностью на запуск заказов нескольким поставщикам, то нужен механизм увязки сообщений-ответов с экземплярами параллельной деятельности.

Получено 23 декабря 2010 г.

Редактор *Е. В. Сабаева*

Подписано в печать 28.06.2011.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 2,44. Уч.-изд. л. 2,93. Тираж 310 экз. Заказ № 57354.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/