Л.К. Птицына Н.Г. Смирнов

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Управление крупно-гранулярными процессами на основе языка BPEL

Учебное пособие

Санкт-Петербург
Издательство Политехнического университета
2011

УДК 004.42:004.438 (075.8)
ББК 32.973-018я73

П 874

Птицына Л.К. Смирнов Н.Г.. Программное обеспечение компьютерных сетей. Управление крупно-гранулярными процессами на основе языка BPEL: Учеб. пособие. СПб.: Изд-во Политехн. ун-та, 2011. – 105 с.

В пособии дана краткая характеристика языка BPEL, описаны история его возникновения, назначение и связь с другими языками программирования. В доступной форме изложены основные положения спецификации языка. Рассмотрены общие представления о структуре и этапах создания BPEL-процесса. Формализован процесс формирования моделей интеграции сервис-ориентированных средств. Описаны методы определения показателей качества интеграции. Раскрыты приемы техники разработки BPEL-процессов для применения в реальных промышленных системах.

Ил. 68. Библиогр.: 15 назв.

Введение

Язык BPEL, также известный как BPEL4WS и WS-BPEL, – это XML подобный язык, основывающийся на спецификации веб-сервисов и использующийся для моделирования и исполнения рабочих процессов. Аббревиатура BPEL расшифровывается как Business Process Execution Language. С момента публикации спецификации языка в 2002 году до сегодняшнего дня язык BPEL получил широкое распространение и завоевал большую популярность, как среди разработчиков, так и среди специалистов в сфере управления.

Перечислим основные причины успешного внедрения языка BPEL:

- Во многих языках программирования манипуляции XML-сообщениями, характерные для сервис-ориентированной архитектуры, являются довольно трудоемкими.
- При использовании классических языков программирования, таких как C# или Java, появляются затруднения в решении задач управления бизнес-процессами в условиях частого изменения бизнес-логики.
- Язык BPEL гармонично встраивается в системы, разрабатываемые в рамках сервис-ориентированной архитектуры. В нем предусматривается большое количество конструкций для работы с веб-сервисами: обработка исключений, корреляция, компенсация и т.д., а также лаконичных и удобных в использовании механизмов манипуляции данными XML-сообщений.
- Разработка бизнес-процессов становится относительно простой задачей благодаря применению редакторов, использующих графическую нотацию BPMN (Business Process Modeling Notation). Достаточно быстро могут создаваться BPEL-процессы, что сводится к удешевлению реализации проектов разработки программных продуктов на основе языка BPEL.

Рабочий процесс в BPEL – это крупно-гранулярный[1] сервис, который выполняет определенные шаги для достижения бизнес-цели и кумулятивно[2] изменяет параметры своего состояния при исполнении

[1] Крупно-гранулярный (large-grained) процесс соответствует решению относительно крупной цельной задачи. Подзадачам соответствуют с мелко-гранулярные процессы.
[2] Кумулятивный – накопленный, интегральный.

запросов клиентов. Цель может представлять собой выполнение бизнес-транзакции либо предоставление какого-либо сервиса. На каждом шаге для достижения цели сервис выполняет действия, являющиеся элементами языка BPEL. В ходе выполнения бизнес-процесса действия вызывают партнерские сервисы, возвращая результат в вызывающий процесс после их выполнения. Подобная совместная работа партнерских сервисов, агрегированная бизнес-процессом, называется оркестровкой сервисов (service orchestration).

Язык BPEL нацелен на решение специфичной, но очень важной задачи – оркестровки сервисов в крупно-гранулярные сервисы или процессы. Грань, по которой отделяют сервисы от процессов, условна. Если оставаться в рамках терминологии сервис-ориентированной архитектуры, оркестровка представляет собой процесс, в котором основной сервис организует совместную работу нескольких других сервисов. Когда для оркестровки используется язык BPEL, то этим основным сервисом является BPEL-процесс, который контролирует последовательность шагов бизнес алгоритма и вызывает вспомогательные сервисы, называемые партнерскими. Сам BPEL-процесс имплементирован как сервис, поэтому может быть вызван, например, из другого бизнес-процесса.

Для осуществления оркестровки BPEL-процесс содержит логику: действия, вызовы, операции присваивания и элементы структурного программирования.

Процесс BPEL может быть синхронным или асинхронным. Обычно в асинхронном режиме BPEL выполняет задачи, решаемые в течение продолжительного отрезка времени. В этом случае акцентируют внимание на том, что BPEL – это процесс. И наоборот, когда BPEL работает в синхронном режиме, его представляют как сервис.

Учебное пособие посвящено представлению языка BPEL и может служить дополнительным средством описания спецификации языка. В пособии отражены главные аспекты отображения конструкций языка BPEL в графической нотации BPMN.

В первой главе пособия изложена информация об истории возникновения языка BPEL, его назначении и занимаемом им месте среди других языков программирования. Указана роль нотации BPMN в ходе разработки процессов BPEL.

Вторая глава включает описание основной концепции языка BPEL и представление простейших языковых конструкций, после изучения которых читатель сможет разрабатывать несложные, но полнофункциональные BPEL-процессы.

В третьей главе описаны ключевые приемы техники разработки BPEL-процессов для применения в реальных промышленных системах. Рассмотрены способы динамического разрешения партнерских связей. Изучение данной главы позволит читателю перейти к построению гибких и масштабных процессов.

В четвертой главе раскрыты способы расширения функциональности языка BPEL за счет введения новых конструкций.

В пятой главе представлена структура технологического процесса создания программного обеспечения, приведены основания для выбора программной платформы, дана характеристика высокоуровневого описания логики бизнес-процесса, формализованы приемы построения модели интеграции сервис-ориентированных средств, указаны методы оценки показателей качества совместной работы служб, приведен вывод аналитических зависимостей показателей от параметров моделей интеграции сервис-ориентированных средств, описаны ключевые особенности кодирования веб-сервиса и разработки бизнес-процесса, а также тестирования программного комплекса.

Термины и определения

BPEL	Business Process Execution Language – язык описания выполнения бизнес-процессов
BPM	Business Process Management – комплекс технологий для разработки приложений, обеспечивающих моделирование и управление бизнес-процессами
BPMN	Business Process Modeling Notation – нотация моделирования бизнес-процессов
BPMI	Business Process Management Initiative – международная организация, разрабатывающая стандарты в области управления бизнес-процессами
OMG	Object Management Group – организация по стандартизации
SOA	Service-Oriented Architecture – сервис-ориентированная архитектура; подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами
SOAP	протокол взаимодействия программных систем с помощью структурированных сообщений в распределенной среде
XML	Extensible Markup Language – расширяемый язык разметки
WSDL	Web Services Description Language – XML-подобный язык описания веб-сервисов, используемый для определения моделей сервисов
Compensation handler	компенсационный обработчик – механизм отката проделанной работы в процессе BPEL
Termination handler	обработчик завершения – часть логики BPEL-процесса, соответствующая завершению определенной области действий
Correlation set	корреляционный набор – совокупность частей входящего сообщения, служащая для уникальной идентификации экземпляра процесса-адресата
Message exchange	взаимосвязь сообщений – механизм для разрешения неоднозначностей при обмене сообщениями с партнерским сервисом
EPR	Endpoint Reference – ссылка точки доступа к веб-сервису

XSLT	XSL Transformations – язык, предназначенный для изменения структуры документа XML или его преобразования в документ на другом диалекте XML
WS-Addressing	Web Services Addressing – спецификация, описывающая транспортно-независимый механизм, позволяющий веб-сервисам взаимодействовать с использованием адресной информации (EPR и др.)
JSR	Java Specification Request – запрос на спецификацию языка Java; документ, описывающий спецификацию технологии, реализованной на платформе Java

1. История языка BPEL и нотация BPMN

1.1. Цель создания языка BPEL

Язык BPEL создан для описания длительных бизнес-процессов, взаимодействующих с веб-сервисами, определенными с использованием языка WSDL. При этом сервисная модель используется в качестве средства композиции процессов в крупно-гранулярные сущности. Язык предусматривает наличие ограниченного набора функций манипуляции данными, достаточного для организации потоков данных и управления.

Полный список задач, поставленных в ходе проектирования спецификации языка BPEL можно найти в [1].

1.2. Связь BPEL с языками программирования

Первая спецификация языка исполнения бизнес-процессов для веб-сервисов (BPEL4WS) вышла в июле 2002 года. Её разработка велась совместными усилиями специалистов из компаний IBM, Microsoft и BEA. Документ, вышедший под версией BPEL4WS 1.0, объединил в себе многие концепции языков программирования WSFL (Web Services Flow Language) от IBM и XLANG от Microsoft.

В мае 2003 года вышла новая версия спецификации – 1.1, к разработке которой присоединились новые специалисты из компаний SAP и Siebel Systems. Указанная версия разработана более тщательно и, кроме того, поддержана многими компаниями на рынке программного обеспечения, создающими программные системы исполнения бизнес-процессов в соответствии с BPEL4WS 1.1. Сразу после этого релиза спецификацию BPEL4WS отправили в стандартизирующий комитет OASIS для утверждения её как официального открытого стандарта, где он был опубликован с изменениями под версией WS-BPEL 2.0.

1.3. Назначение нотации

Нотация моделирования рабочих процессов BPMN (Business Process Modeling Notation) разработана организацией BPMI (Business Process Management Initiative), которая выполняет функции по её сопровождению. Основной целью BPMN является предоставление системы обозначений, понятной и легко воспринимаемой всеми специалистами, задействованными в создании и управлении рабочими процессами предприятия, начиная от аналитиков, проектирующих процессы, программистов, имплементирующих процессы с помощью технологий, позволяющих их исполнять, и заканчивая менеджерами, ответственными за управление и мониторинг. Таким образом, BPMN служит для того, чтобы заполнить разрыв между проектированием рабочих процессов и их технической имплементацией.

Другой, не менее значимой, целью BPMN является необходимость удостовериться в том, что XML-подобные языки исполнения процессов, такие как, например, BPEL4WS, могут быть визуализированы при помощи бизнес ориентированной нотации.

BPMN – это графическая нотация. Последнее означает, что её ключевой составляющей является набор графических элементов, служащих для обозначения различных объектов, специфичных для управления рабочими процессами. Следует иметь в виду, что при применении этих графических элементов их внешний вид (размер, цвет, вид линий и расположение подписей) может быть изменен в тех случаях, когда это не определяется спецификацией явным образом.

Следует также отметить, что не все процессы оркестровки, описанные в нотации BPMN 2.0, могут быть сопоставлены простым образом рабочим процессам BPEL4WS. Данное обстоятельство является следствием того, что нотация BPMN позволяет строить графы почти любой сложности для моделирования рабочего процесса, тогда как в языке BPEL существуют определенные ограничения для моделируемых процессов, обусловленные технической составляющей. В данной работе больший интерес представляет обратное представление – из WS-BPEL в BPMN, поэтому проблемы прямого представления из BPMN в WS-BPEL не коснутся дальнейшего изложения.

Участие в разработке BPMN представителей многих крупных компаний, использующих различные нотации описания рабочих процессов, позволило объединить опыт и их лучшие качества в BPMN. Примерами нотаций и методологий, рассмотренных в ходе разработки, являются UML Activity Diagram, UML EDOC Business Processes, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, and Event-Process Chains (EPCs).

Как указано в спецификации нотации BPMN 2.0, разработчики программного обеспечения создания и редактирования BPEL-процессов с поддержкой графического моделирования могут отходить от строгого следования проформам изображения элементов нотации в определенных границах. В связи с этим наряду и вместо некоторых изображений из спецификации в учебное пособие включены изображения элементов BPEL, используемых в среде разработки NetBeans 6.5, что позволит читателю ориентироваться в аналогичных обозначениях элементов, встречающихся в одном из наиболее популярных BPEL-редакторов.

2. Простейшие конструкции

2.1. Структура BPEL-процесса

Бизнес-процесс в терминах языка BPEL – это контейнер, где можно объявить связи с внешними партнерами, данные процесса, обработчики для различных целей, и, самое главное, исполнимые действия процесса. Процесс обладает рядом атрибутов таких, как, например, имя и пространство имен (обязательные атрибуты). Простейший вариант объявления BPEL-процесса приведен в листинге 1.

```
<process name="PrimerProcess"
   targetNamespace="http://oasis-open.org/WSBPEL/Primer/"
   xmlns="http://docs.oasis-
open.org/wsbpel/2.0/process/executable" />
```

Листинг 1. Пример BPEL-процесса [2]

Декларация пространства имен "http://docs.oasis-open.org/wsbpel/2.0/process/executable" указывает на то, что процесс исполняемый. Существуют и неисполняемые процессы, также называемые абстрактными. Они частично описывают поведение процесса, не уточняя все детали логики. Используются обычно в качестве шаблонов или для объявления интерфейса, необходимого партнерам, без явного открытия внутренней бизнес логики. Исполняемый процесс, напротив, полностью определяет поведение, а также объявляет часть, требующуюся партнерам. Графические представления процесса в нотации BPMN и в редакторе NetBeans представлены на рис.1 и рис.2.

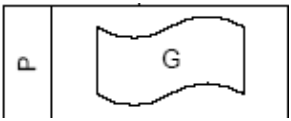


Рис. 1. Изображение процесса в нотации BPMN [3]

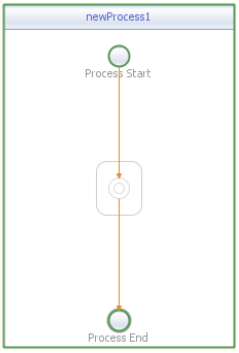


Рис. 2. Изображение процесса в BPEL-редакторе среды разработки NetBeans 6.5

Корневой элемент файла BPEL-процесса – process. Он является контейнером для объявления связей с партнерами, данных и обработчиков, считающихся глобальными для всего процесса. Язык BPEL также позволяет делать такие объявления локальными, для этого существует элемент scope (листинг 2, рис.3), задающий область видимости объявленным сущностям.

```
<scope name="Scope" />
```

Листинг 2. Пример элемента Scope

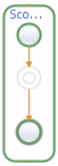


Рис. 3. Изображение Scope в среде NetBeans

С помощью элемента scope можно разделить логику BPEL-процесса на несколько независимых частей. Переменные и другие сущности элемента scope будут видны только в данном элементе и не будут доступны другим частям процесса.

2.2. Взаимосвязь с партнерами процесса

Процесс BPEL позволяет агрегировать функциональность веб-сервисов и определять бизнес-логику между операциями взаимодействия с ними. За это его называют ещё процессом оркестровки (интеграции) веб-сервисов. Любое взаимодействие BPEL с веб-сервисом можно трактовать как обращение к бизнес-партнеру. Это взаимодействие описывается с помощью так называемых партнерских связей (partnerLink). PartnerLink – это конструкция BPEL, своего рода типизированный коннектор, определяющий, какие WSDL порты процесс предоставляет бизнес партнерам и какие требует от них.

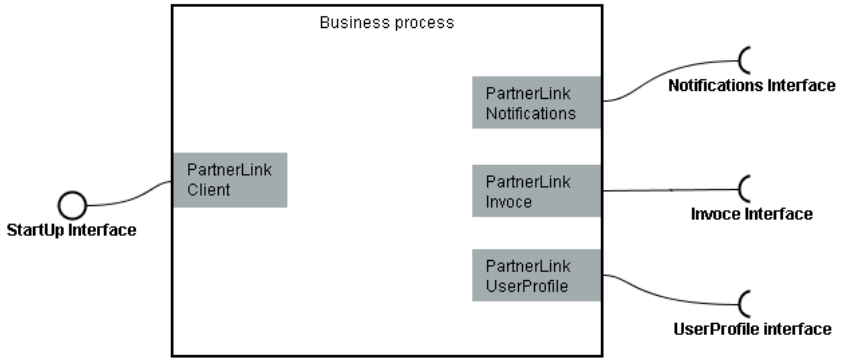


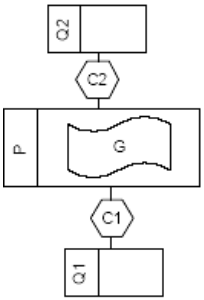
Рис. 4. Партнерские сервисы BPEL-процесса

Для каждого партнера может быть несколько партнерских связей (partnerLink). Можно представить объект partnerLink как коммуникационный канал между партнером и BPEL-процессом, как это представлено на рис.4. Подобное взаимодействие может быть двунаправленным: процесс вызывает партнера, партнер вызывает процесс. Таким образом, каждый объект partnerLink характеризуется типом и ролью (см. листинг 3). Данная информация определяет интерфейсы взаимодействия, один из которых должен предоставлять процесс, а второй – партнерский сервис.

```
<partnerLinks>
   <partnerLink name="ClientStartUpLink"
      partnerLinkType="wsdl:ClientStartUpPLT" myRole="Client" />
</partnerLinks>
```

Листинг 3. Партнерская связь (partnerLink) [2]

Графическое представление партнерских связей представлено на рис.5 и рис.6.



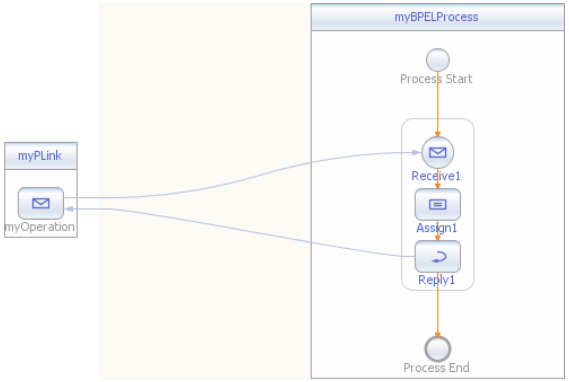


Рис. 5. Изображение partnerLink в нотации BPMN [3]

Рис. 6. Изображение partnerLink в среде NetBeans

Объявление партнерских связей может следовать непосредственно внутри элемента process, в этом случае они будут доступны всем частям процесса, либо внутри элемента scope – тогда они будут видны только дочерним элементам данной области.

2.3. Состояние BPEL-процесса

Выше было использовано понятие данных BPEL-процесса. Технически это понятие можно определить как набор переменных, объявленных в элементах process или scope. Совокупность данных, присвоенных этим переменным, является состоянием процесса во время исполнения. Переменные в языке BPEL типизированы. Они могут

содержать данные либо полученные от партнерских сервисов, либо созданные и используемые внутри самого процесса. Для того, чтобы данные процесса были совместимы с интерфейсами партнерских сервисов, переменные могут быть либо типами сообщений WSDL, либо простыми или сложными типами XML схемы, либо элементами XML схемы. Изменение состояния BPEL-процесса происходит при изменении значений переменных, объявленных в нем. По умолчанию BPEL использует язык XPath 1.0 для доступа к переменным и для их изменения.

Для того, чтобы объявить переменную, необходимо указать её имя и тип, один из трех указанных выше. Имя указывается при помощи атрибута name, а тип соответственно при помощи атрибутов messageType, type либо element, как представлено в листинге 4.

```
<variables>
   <variable name="myVar1"
             messageType="myNS:myWSDLMessageDataType" />
   <variable name="myVar2" type="xsd:string" />
   <variable name="myVar3" type="myNS:myComplexType" />
   <variable name="myVar4" element="myNS:myXMLElement" />
</variables>
```

Листинг 4. Пример объявления переменных [2]

Переменные могут быть объявлены непосредственно внутри элемента process, в таком случае они будут видны всем частям процесса, либо внутри элемента scope, тогда они будут локальными для данной области.

2.4. Поведение BPEL-процесса

Основными элементами, из которых конструируется BPEL-процесс, являются действия (activities). Существует два типа действий:

- структурные, они могут содержать другие действия и связывать их, определяя бизнес логику процесса;
- элементарные, они выполняют только закрепленную за ними определенную функциональность (например, принимать сообщение от партнера или изменять данные) и не содержат другие действия.

2.4.1. Получение и передача сообщений веб-сервисов

В языке BPEL существует несколько действий, предназначенных для получения и передачи сообщений веб-сервисов. Это receive, reply, invoke действия. Все они позволяют обмениваться сообщениями с внешними партнерскими сервисами.

Назначение действия receive состоит в получении сообщения от внешнего партнера. Для этого действия необходимо указать партнерскую связь (partnerLink) и имя вызываемой операции. При этом необходимо обозначить переменную (или набор переменных), которая будет содержать данные, полученные во входящем сообщении от партнера. Если выбранная операция двунаправленная, то есть требующая ответного сообщения от процесса партнеру, то действию receive в алгоритме должно соответствовать действие reply. Пример объявления действия receive приведен в листинге 5.

```
<receive name="ReceiveRequestFromPartner"
   createInstance="yes"
   partnerLink="ClientStartUpPLT"
   operation="StartProcess" ... />
```

Листинг 5. Получение сообщения действием receive [2]

Атрибут createInstance="yes" означает, что действие receive создаст и запустит новый экземпляр BPEL-процесса, тогда как "no" будет означать, что сообщение будет получено в уже запущенный экземпляр процесса. Графический вид данного действия приведен на рис.7 и рис.8.

Рис. 7. Событие получения сообщения в нотации BPMN [3]

Рис. 8. Событие получения сообщения в среде NetBeans

Как было указано выше, каждое действие receive может иметь соответствующее действие reply. Примером такой ситуации может служить процесс заказа книги в библиотеке. Клиент отправит сообщение действию receive этого процесса, процесс, в свою очередь, выполнит бизнес логику заказа (проверку введенных данных клиента, проверку наличия книги и т.п.) и после её завершения ответит клиенту действием

reply с отправкой сообщения. Действие reply может посылать сообщения двух типов:

- нормального, такие сообщения посылаются в ходе нормального исполнения логики;
- аварийного, такие сообщения формируются в результате возникновения каких-либо исключительных (ошибочных) событий (например, при отсутствии в библиотеке свободного экземпляра заказываемой клиентом книги). В этом случае в действии reply необходимо указать атрибут faultName.

Как правило, действие reply применяется в паре с действием receive для имплементации WSDL операций типа запрос-ответ в конкретном коммуникационном канале (partnerLink). Для выбора канала, через который был получен запрос, в reply указывается атрибут partnerLink и имя операции в атрибуте operation (см. листинг 6, рис. 9). При этом указывается переменная, которая содержит ответ для клиента – нормальный либо аварийный. Если возвращается аварийный ответ, то действие reply должно содержать атрибут faultName, ссылающийся на соответствующее сообщение об ошибке, описанное в WSDL.

```
<reply name="ReplyResponseToPartner"
   partnerLink="ClientStartUpPLT"
   operation="StartProcess" ... />
```

Листинг 6. Действие reply [2]

Рис. 9. Действие reply в среде NetBeans

Третьим действием, относящимся к взаимодействию с веб-сервисами в BPEL, является invoke. Действие invoke служит для вызова партнерского веб-сервиса. Элемент invoke должен содержать коммуникационный канал (partnerLink) и имя вызываемой операции (см. листинг 7).

```
<invoke name="InvokePartnerWebService"
   partnerLink="BusinessPartnerServiceLink"
   operation="partnerOperation" ... />
```

Листинг 7. Действие invoke [2]

Графический вид данного действия приведен на рис. 10.

Рис. 10. Действие invoke в среде NetBeans

В спецификации языка WSDL версии 1.1 описаны несколько типов операций. Два из них поддержаны спецификацией BPEL: однонаправленные и запрос-ответ. Действие invoke может либо послать запрос к однонаправленной операции (и продолжить дальнейшее выполнение логики процесса без ожидания ответа от партнерского сервиса), либо отправить сообщение к операции типа запрос-ответ (тогда выполнение логики процесса приостановится до получения ответа от вызываемого партнерского сервиса). В последнем случае элемент invoke должен указывать переменную, содержащую данные для запроса, а также переменную, предназначенную для хранения данных, вернувшихся в ответном сообщении от партнерского сервиса. Листинг 8 содержит пример объявления действия invoke.

```
<invoke name="RequestResponseInvoke"
   partnerLink="BusinessPartnerServiceLink"
   operation="RequestResponseOperation"
   inputVariable="Input"
   outputVariable="Output" />
```

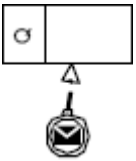
Листинг 8. Вызов операции типа запрос-ответ действием invoke [2]

В случае вызова однонаправленной операции необходимо указать только переменную с запросом, как указано в листинге 9.

```
<invoke name="OneWayInvoke"
   partnerLink="BusinessPartnerServiceLink"
   operation="OneWayOperation"
   inputVariable="Input" />
```

Листинг 9. Вызов однонаправленной операции действием invoke [2]

На рис. 11 и рис. 12 изображены графические представления элемента invoke.



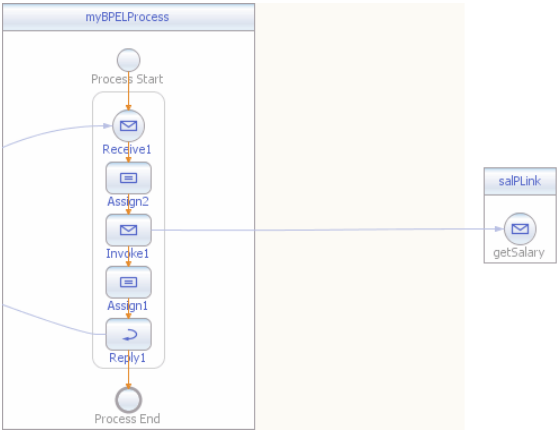


Рис. 11. Изображение операции вызова в нотации BPMN [3]

Рис. 12. Изображение операции вызова в среде NetBeans

Кроме того, можно связать выполнение действия invoke с различными обработчиками, как это делается для элементов process и scope. Вообще можно трактовать действие invoke как сокращенное обозначение структурного действия scope с объявленным внутри него элементом invoke. Упомянутые обработчики действия invoke могут быть двух типов:

- обработчики ошибок;
- компенсационные обработчики.

Далее мы рассмотрим более подробно эти типы.

Для полноты описания стоит отметить, что в BPEL существует ещё несколько конструкций, связанных с взаимодействием процесса и веб-сервисов. Это действие pick и обработчик событий, которые будут описаны подробнее ниже.

2.4.2. Структурные языковые конструкции

Реализация логики процесса на языке BPEL не обходится без элементов структурного программирования. Если нужно описать ряд последовательно выполняющихся действий (например, получение заказа, проверка оплаты, поставка товара, подтверждение поставки), то для этого

можно использовать элемент sequence языка BPEL. Иными словами, действие sequence используется для определения набора действий, исполняемых последовательно в лексическом порядке, как, например, в листинге 10.

```
<sequence name="InvertMessageOrder">
   <receive name="receiveOrder" ... />
   <invoke name="checkPayment" ... />
   <invoke name="shippingService" ... />
   <reply name="sendConfirmation" ... />
</sequence>
```

Листинг 10. Действие sequence [2]

Последовательность действий изображена на рис. 13 и рис. 14.

Рис. 13. Последовательность sequence в нотации BPMN [3]

Рис. 14. Последовательность sequence в среде NetBeans

Конструкция if-else является ещё одним инструментом для структурирования логики процесса. Эта конструкция знакома всем по классическим языкам программирования. Действие if-else позволяет выбрать единственную ветвь исполнения среди данного набора. Работа if-else элемента сводится к проверке условия каждой ветви процесса, и если это условие верно, то к исполнению данной ветви, иначе – к переходу на следующую ветвь. Выражение, описывающее условие перехода по любой ветви, может использовать язык XPath. Следует обратить внимание, что только первая ветвь с верным условием будет исполнена. Если ни одно условие не принимает значения true, то, в случае присутствия, выполнится альтернативная ветвь, определенная элементом else (см. листинг 11).

```
<if name="isOrderBiggerThan5000Dollars">
   <condition>
      $order &gt; 5000
   </condition>
   <invoke name="calculateTenPercentDiscount" ... />
   <elseif>
      <condition>
         $order &gt; 2500
      </condition>
      <invoke name="calculateFivePercentDiscount" ... />
   </elseif>
   <else>
      <reply name="sendNoDiscountInformation" ... />
   </else>
</if>
```

Листинг 11. Действие if-else [2]

Графическое представление условного ветвления изображено на рис. 15 и рис. 16.

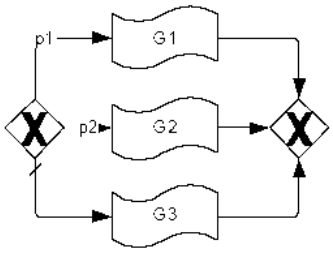


Рис. 15. Условное ветвление в нотации BPMN [3]

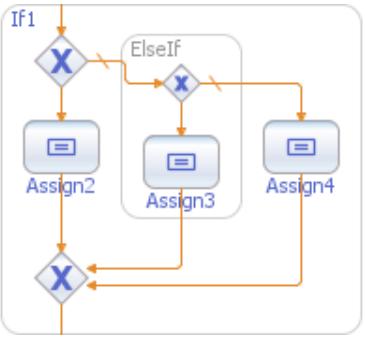


Рис. 16. Условное ветвление в среде NetBeans

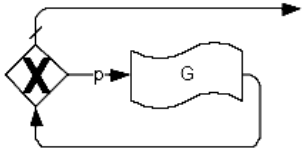
2.4.3. Циклические действия

Язык BPEL предоставляет три действия, с помощью которых можно многократно повторять часть логики процесса. Одним из них является действие while. Это структурное действие, то есть оно имеет

вложенный дочерний элемент и позволяет повторять его исполнение столько раз, пока условие цикла будет иметь значение true. Условие контролируется перед каждой итерацией цикла, из чего следует, что дочерний элемент может быть не выполнен ни разу (если при контроле устанавливается значение false). Пример цикла while приведен в листинге 12, а его графическая нотация на рис. 17 и рис. 18.

```
<while>
   <condition>
      $iterations &gt; 3
   </condition>
   <invoke name="increaseIterationCounter" ... />
</while>
```

Листинг 12. Действие while [2]



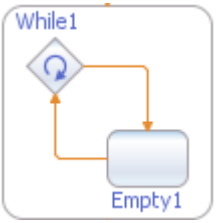


Рис. 17. While в нотации BPMN [3] **Рис. 18. While в среде NetBeans**

Действие repeatUntil отличается от while тем, что его дочерний элемент будет выполнен хотя бы один раз, так как условие цикла контролируется после исполнения каждой итерации, как указано в листинге 13 и следует из графического представления на рис. 19 и рис. 20.

```
<repeatUntil>
   <invoke name="increaseIterationCounter" ... />
   <condition>
      $iterations &gt; 3
   </condition>
</repeatUntil>
```

Листинг 13. Действие repeatUntil [2]

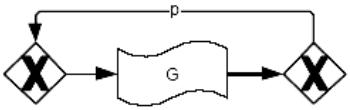


Рис. 19. RepeatUntil в нотации BPMN [3]

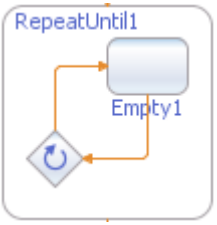


Рис. 20. RepeatUntil в среде NetBeans

Ещё одним элементом для циклического исполнения является действие forEach. По умолчанию его поведение сводится к последовательному исполнению дочернего элемента scope N раз. Примером использования такого поведения может служить перебор содержимого сообщения заказа, состоящего из N позиций. Технически действие forEach выполняет дочернее действие scope N раз, где

N = finalCounterValue – startCounterValue + 1.

Пример объявления цикла forEach представлен в листинге 14.

```
<forEach parallel="no" counterName="N" ...>
   <startCounterValue>1</startCounterValue>
   <finalCounterValue>5</finalCounterValue>
   <scope>
      <documentation>
         check availability of each item ordered
      </documentation>
      <invoke name="checkAvailability" ... />
   </scope>
</forEach>
```

Листинг 14. Действие forEach (последовательный вариант) [2]

Графические представления цикла forEach с помощью нотации BPMN и в среде разработки NetBeans изображены рис. 21 и рис. 22.

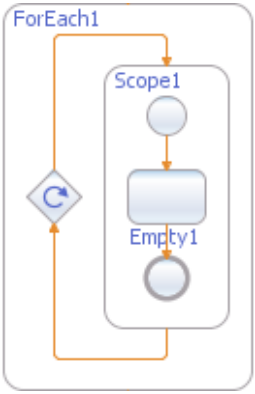


Рис. 21. forEach в нотации BPMN [3] **Рис. 22. forEach в среде NetBeans**

Существует два варианта исполнения действия forEach – последовательный и параллельный, что указывает атрибут parallel. Последний вариант рассмотрен более подробно далее.

В отличие от других структурных действий, рассмотренных выше, forEach имеет одно ограничение. Тогда как остальные подобные действия могут иметь в качестве дочернего элемента произвольное действие, forEach может содержать только действие scope.

2.4.4. Параллельные вычисления

Возможности языка BPEL, связанные со структурой процессов в последовательном виде, достаточны для создания широкого класса систем, полезных для практического применения. Однако бывают ситуации, когда параллельное исполнение некоторых частей процесса бывает не просто выгодным, но и необходимым условием для успешного завершения работы. Для распараллеливания выполнения отдельных частей процесса в языке BPEL существует действие flow. Параллельного исполнения можно также добиться посредством использования параллельного варианта действия forEach и обработчика событий. Указанные методы будут рассмотрены ниже.

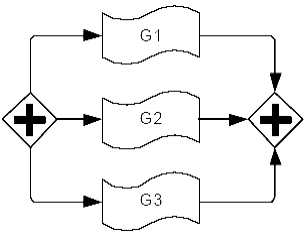
В следующем примере три действия (checkFlight, checkHotel и checkRentalCar) выполняются параллельно, то есть соответствующие веб-

сервисы вызываются асинхронно (листинг 15). Все три действия стартуют независимо, когда действие flow получает управление.

```
<flow ...>
   <links> ... </links>
   <documentation>
      check availability of a flight, hotel and rental car
concurrently
   </documentation>
   <invoke name="checkFlight" ... />
   <invoke name="checkHotel" ... />
   <invoke name="checkRentalCar" ... />
</flow>
```

Листинг 15. Действие flow [2]

Графическая нотация параллельных действий приведена рис. 23 и рис. 24.



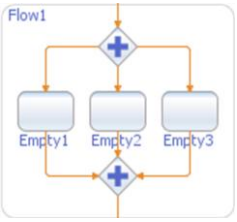


Рис. 23. Flow в нотации BPMN [3] **Рис. 24. Flow в среде NetBeans**

Иногда работа независимых частей процесса, выполняемых параллельно, требует синхронизации. Представим, что кроме упомянутых в примере параллельных действий, процесс включает в себя вызов четвертого действия bookFlight. Если просто добавить это действие к предыдущим трем, то оно так же как остальные начнет своё независимое исполнение сразу после старта действия flow. Но в рассматриваемом примере действие bookFlight имеет смысл только, если успешно завершится действие checkFlight. Таким образом, можно указать на эту зависимость, добавив связь (элемент link) между этими двумя действиями. Добавление такой связи приводит к введению управляющей зависимости, означающей, что действие, отмеченное как цель связи (target), будет выполнено только в том случае, если действие, отмеченное как источник связи (source), завершится. Соответствующий код BPEL приведен в листинге 16, а его графическое представление на рис. 25.

```
<flow ...>
   <links>
      <link name="checkFlight-To-BookFlight" />
   </links>
   <documentation>check availability of a flight, hotel and rental
car concurrently</documentation>
   <invoke name="checkFlight" ...>
      <sources>
         <source linkName="checkFlight-To-BookFlight" />
      </sources>
   </invoke>
   <invoke name="checkHotel" ... />
   <invoke name="checkRentalCar" ... />
   <invoke name="bookFlight" ...>
      <targets>
         <target linkName="checkFlight-To-BookFlight" />
      </targets>
   </invoke>
</flow>
```

Листинг 16. Действие flow с синхронизирующей связью [2]

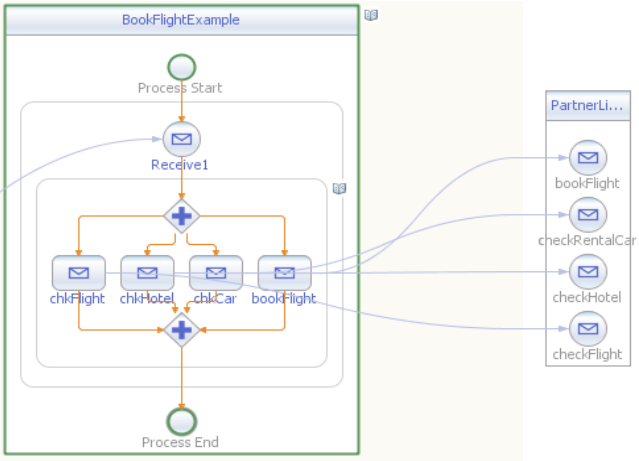


Рис. 25. Пример BPEL-процесса заказа билета на самолет с использованием действия flow в среде NetBeans

Семантика элемента link гораздо шире, чем показано в этом небольшом примере. Связь (link) может иметь транзитивное условие, которое определяет статус связи. Если transitionCondition не определено

явно, то статус связи равен true, иначе transitionCondition будет определять этот статус. Рассмотрим пример в листинге 17.

```
<flow ...>

   <links>
      <link name="request-to-approve" />
      <link name="request-to-decline" />
   </links>
   <receive
      name="ReceiveCreditRequest"
      createInstance="yes"
      partnerLink="creditRequestPLT"
      operation="creditRequest"
      variable="creditVariable">
      <sources>
         <source linkName="request-to-approve">
            <transitionCondition>
               $creditVariable/value &lt; 5000
            </transitionCondition>
         </source>
         <source linkName="request-to-decline">
            <transitionCondition>
               $creditVariable/value >= 5000
            </transitionCondition>
         </source>
      </sources>
   </receive>
   <invoke name="approveCredit" ...>
      <targets>
         <target linkName="request-to-approve" />
      </targets>
   </invoke>
   <invoke name="declineCredit" ...>
      <targets>
         <target linkName="request-to-decline" />
      </targets>
   </invoke>

</flow>
```

Листинг 17. Действие flow со связью и транзитивным условием [2]

Графическое представление фрагмента BPEL-процесса из листинга 17 приведено на рис. 26.

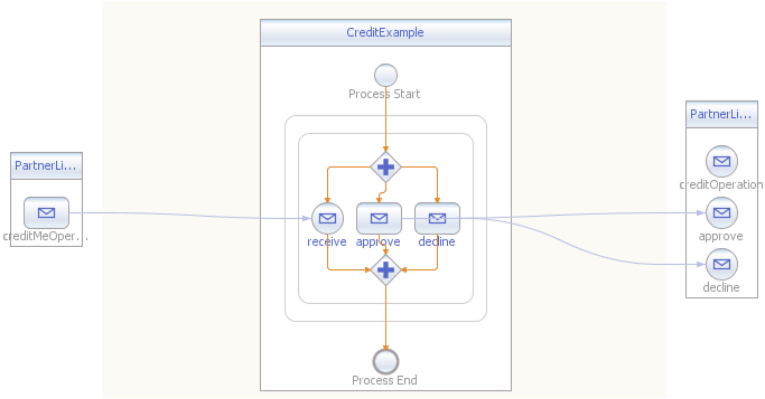


Рис. 26. Пример BPEL-процесса кредитования с использованием действия flow в среде NetBeans

Связь request-to-approve ассоциирована с транзитивным условием, проверяющим то, что элемент value, входящий в состав содержимого переменной creditVariable, меньше, чем 5000. Если в ходе исполнения процесса такое условие будет верно, то статус связи request-to-approve примет значение true, а действие, являющееся целью связи, будет выполнено. Иначе, статус примет значение false и действие target выполнено не будет.

Так как транзитивное условие связи request-to-decline логически противоположно (больше или равно 5000) условию, ассоциированному с request-to-approve, то это означает, что будет выполнено только одно из действий approveCredit и declineCredit.

Транзитивные условия предоставляют разработчику механизм для разделения основного потока исполнения на несколько других потоков, основываясь на значениях этих условий. Последнее означает, что должен быть способ соединения этих потоков в один. Язык BPEL решает эту проблему с помощью объединяющих условий (joinCondition). Подобные условия ассоциированы с действиями, как правило, имеющими входящие связи. Элемент joinCondition определяет для ассоциированного с ними действия подобие "стартового условия", то есть для того, чтобы действие стартовало, необходимо, чтобы все входящие связи имели статус true, либо хотя бы одна входящая связь имела статус true. Проиллюстрируем это на примере в листинге 18 и на соответствующем ему рис. 27.

```
<flow ...>
   <links>
      <link name="request-to-approve" />
      <link name="request-to-decline" />
      <link name="approve-to-notify" />
      <link name="decline-to-notify" />
   </links>
   <receive name="ReceiveCreditRequest"
      createInstance="yes"
      partnerLink="creditRequestPLT"
      operation="creditRequest"
      variable="creditVariable">
      <sources>
         <source linkName="request-to-approve">
            <transitionCondition>
               $creditVariable/value &lt; 5000
            </transitionCondition>
         </source>
         <source linkName="request-to-decline">
            <transitionCondition>
               $creditVariable/value >= 5000
            </transitionCondition>
         </source>
      </sources>
   </receive>
   <invoke name="approveCredit" ...>
      <source linkName="approve-to-notify" />
      <targets>
         <target linkName="request-to-approve" />
      </targets>
   </invoke>
   <invoke name="declineCredit" ...>
      <source linkName="approve-to-notify" />
      <targets>
         <target linkName="request-to-decline" />
      </targets>
   </invoke>
   <reply name="notifyApplicant" ...>
      <targets>
         <joinCondition>
            $approve-to-notify or $decline-to-notify
         </joinCondition>
         <target linkName="approve-to-notify" />
         <target linkName="decline-to-notify" />
      </targets>
   </reply>
</flow>
```

Листинг 18. Действие flow с объединяющими условиями [2]

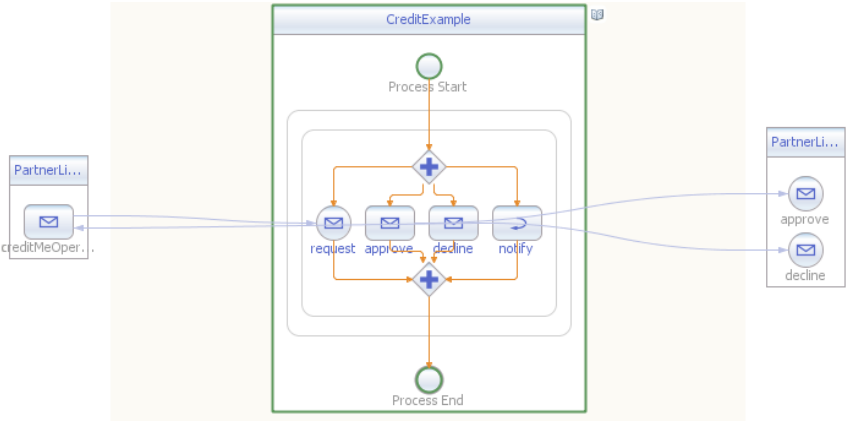


Рис. 27. Пример BPEL-процесса кредитования с использованием действия flow с объединением в среде NetBeans

Представим себе, что в действии approveCredit возникла ошибка, его исходящая связь apprive-to-notify будет установлена средой исполнения в значение "false". Подобное может привести к ситуации, когда joinCondition также будет иметь значение "false". Язык BPEL предоставляет два способа обработки объединяющих условий со значением false. Во-первых, в таких ситуациях по умолчанию возникает исключение joinFault, которое можно обработать соответствующим обработчиком ошибок (см. более подробно ниже). Во-вторых, когда у процесса или scope атрибут suppressJoinFailure выставлен в "yes", то действие, ассоциированное с объединяющим условием со значением false, пропускается, а false-статус связи передается всем исходящим из этого действия связям. Иными словами, false-статус связи будет транзитивно распределен среди остальных путей, образованных следующими связями, до тех пор, пока не будет достигнуто такое объединяющее условие, которое примет значение true. Представленный подход называется исключением мертвых путей (Dead-Path Elimination, DPE).

2.4.5. Обработка данных

В предыдущих параграфах дано описание языковых конструкций BPEL-процесса, позволяющих получать и отправлять данные. На практике часто возникает необходимость разделить эти данные на различные части или объединить в единое целое из разных источников.

Допустим, что существует входящее сообщение, состоящее из нескольких частей:

- имя заказчика;
- номер заказанного товара;
- адрес доставки;
- информация о кредитной карточке (номер, дата завершения обслуживания карты и т.п.).

Теперь представим, что компания устроена так, что один отдел занимается доставкой товаров, другой следит за присутствием товаров на складе, а проверка кредитных карт осуществляется с помощью партнерской фирмы. Легко догадаться, что партнерский сервис, проверяющий кредитки, мало заинтересован в информации о том, какой товар заказал клиент в фирме. Для работы ему необходимы только имя клиента и информация о карте оплаты. Процесс BPEL должен выделить необходимые две части из входящего сообщения, а также объединить их в новое сообщение, чтобы послать его сервису creditCardCheck. Аналогичный пример можно привести относительно отдела доставки. Становится очевидным, что BPEL-процессу необходим механизм манипуляции данными для решения подобного рода задач.

Рассмотрим действие assign (листинг 19 и рис. 28). Его объявление в BPEL содержит одну или более операций copy. Каждый элемент копирования в свою очередь содержит две части from и to, определяющие источник и приемник операции соответственно. В следующем примере переменная копируется полностью в другую переменную такого же типа.

```
<assign>
   <copy>
      <from variable="TimesheetSubmissionFailedMessage" />
      <to variable="EmployeeNotificationMessage" />
   </copy>
</assign>
```

Листинг 19. Действие assign [2]

Рис. 28. Действие assign в среде NetBeans

Приведенный пример, очевидно, не решает проблему манипуляции данными в полном объеме. Часто бывает нужно копировать только определенные части переменной в другую переменную или даже в определенную часть другой переменной. Следовательно, необходим способ описания доступа к различным частям данных. Одной из целей проектирования BPEL было избежание создания нового XML-подобного языка манипуляции данными и использование существующих стандартов XPath и XSLT для решения этой задачи.

```
<assign>
   <copy>
      <from variable="Input" part="operation1Parameter">
         <query>
            creditCardInformation
         </query>
      </from>
      <to variable="CreditCardServiceInput" />
   </copy>
</assign>
```

Листинг 20. Действие assign с атрибутом query [2]

Как видно из примера в листинге 20, когда необходимо адресоваться к части переменной, можно использовать для этого элемент языка запросов query, в котором прописывается выражение на языке XPath. По умолчанию WS-BPEL 2.0 использует стандарт языка XPath 1.0, версию которого можно указать явным образом в атрибуте queryLanguage элемента query.

Можно присваивать не только переменные или их части другим переменным (и их частям), но допустимы и следующие конструкции:

свойство переменной:

```
<from>
  bpel:getVariableProperty("Input", "a:orderNo")
</from>
```

партнерская связь:

```
<from partnerLink="Supplier"/>
```

операция XPath:

```
<from>count($po/poline)</from>
```

2.4.6. Обработка исключений

Почти весь материал, изложенный выше, описывал работу BPEL в нормальном режиме, то есть когда действия выполняются без ошибок. На

практике такой язык как BPEL должен иметь возможность обработки исключительных ситуаций, например, вызова веб-сервиса, который в данный момент недоступен. Иными словами должны быть доступны механизмы отслеживания исключительных ситуаций и их обработки.

Для этих целей BPEL предлагает использование концепции обработчиков ошибок (исключений). Обработчик ошибок может быть присоединен к элементам scope, process или, как было упомянуто выше, непосредственно к элементу invoke, который можно считать сокращенным вариантом scope с одним действием invoke внутри. В этой части мы рассмотрим обработку ошибок на уровне process.

Обработчики ошибок инициализируются и становятся готовыми к работе сразу после того, как стартует scope, к которому они присоединены. Таким образом, обработчики ошибок элемента process инициализируются после запуска процесса. Если процесс завершился успешно, тогда присоединенные обработчики прекратят свою работу, но в случае, если в ходе исполнения возникло исключение, оно передается соответствующему обработчику.

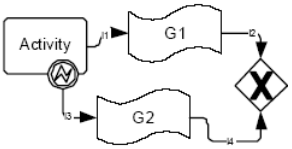
```
<faultHandlers>
   <catch faultName="BookOutOfStockException"
      faultVariable="BookOutOfStockVariable">
      ...
   </catch>
   <catchAll>...</catchAll>
</faultHandlers>
```

Листинг 21. Обработчик ошибок процесса [2]

Как видно из примера в листинге 21, элемент обработчика ошибок может иметь два вида дочерних элементов: один или более catch-блоков и не более одного catchAll-блока. Каждый дочерний элемент должен содержать какое-либо действие (в примере обозначено "…"), которое производит непосредственную обработку ошибки заданного типа. В приведенном примере процесс вызывает веб-сервис checkAvailability, чтобы проверить, доступна ли заказываемая книга. В случае отсутствия необходимой книги, сервис ответит исключением BookOutOfStockException (которое должно быть предварительно объявлено в wsdl интерфейсе сервиса).

Конструкция catch предусматривает наличие дополнительных необязательных атрибутов: faultName, ссылающийся на имя исключения, которое необходимо отловить, и faultVariable. Если указан faultVariable, то вместе с ним должен присутствовать один из атрибутов: faultMessageType или faultElement. Атрибут faultVariable указывает на переменную,

локально определенную в конструкции catch с учетом значений атрибутов faultMessageType или faultElement.



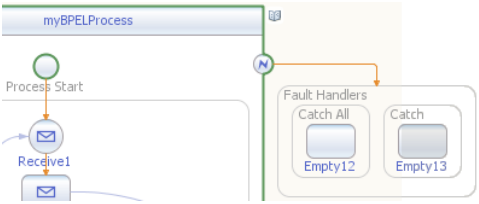


Рис. 29. Обработчик ошибок в нотации BPMN [3]

Рис. 30. Обработчик ошибок в среде NetBeans

Графическая нотация использования обработчиков ошибок представлена на рис. 29 и рис. 30.

Обработчик ошибок может иметь не более одного catchAll-блока. Его цель – обработка исключений, не специфицированных определенным именем, например, если бы веб-сервис checkAvailability бросал ещё два исключения BookOutOfPrintException и BookTitleNotFoundException, кроме упомянутого BookOutOfStockException, и не требовалось бы обрабатывать их раздельно, то можно было бы обработать оба исключения, определив catchAll-блок в конце обработчика ошибок.

3. Конструкции для развитых спецификаций

3.1. Улучшение структуры процесса

Язык BPEL позволяет составлять процесс в виде иерархически вложенных структур scope (область действия). Каждая область действия может определять переменные, партнерские связи, взаимосвязи сообщений (messageExchange), корреляционные наборы (correlationSet) и обработчики (handler) – все они будут доступны для данной области и для всех вложенных областей. Перечисленные сущности также называются контекстом исполнения области действия. Объявление BPEL-процесса является внешним, то есть родительским, для всех областей действия.

Две конструкции языка BPEL включают в себя обязательное использование scope: это действие onEvent обработчика событий и действие forEach. Указанные структуры содержат, соответственно,

определения переменной обработчика событий и переменной счетчика цикла. В случае обработчика событий его дочерняя область действия может также содержать объявление партнерской связи, взаимосвязи сообщений или набора корреляции, используемых обработчиком.

3.1.1. Жизненный цикл области действия

Структура scope может быть использована как обычное действие в языке BPEL, например, в качестве дочернего элемента цикла. Жизненный цикл области действия начинается со следующей инициализирующей последовательности сущностей, локально определенных в данном элементе scope:

- инициализация переменных и партнерских связей;
- инстанцирование корреляционных наборов;
- установка обработчиков ошибок, обработчика завершения и обработчиков событий.

Перечисленные выше шаги выполняются в транзакционном режиме, то есть либо все шаги выполняются успешно и жизненный цикл продолжается, либо исключение bpel:scopeInitializationFailure выбрасывается для обработки родительской областью действия. После инициализации области действия и перехода к стартовому действию её обработчики событий переходят в активное состояние в параллельном режиме по отношению друг к другу и к основному потоку исполнения. Непосредственно после этого начинается процесс выполнения основного действия в данной области.

Scope может завершиться либо успешно, либо неуспешно. Существует три варианта развития событий во время исполнения.

- **Нормальное завершение.** Если основное действие области не выбросило исключений и на все входные сообщения были отправлены необходимые ответные сообщения, тогда все обработчики событий переводятся в неактивный режим (уже запущенные экземпляры обработчиков завершают свою работу в обычном порядке без прерываний), компенсационный обработчик инициализируется и исполняется. Область действия завершается успешно.
- **Внутренняя ошибка.** Если во время исполнения scope выбрасывается исключение, тогда все находящиеся внутри области и запущенные действия и экземпляры обработчиков событий терминируются,

вызывается исполнение подходящего обработчика ошибок. Область действия завершается неуспешно.

- **Внешнее завершение.** Если выполняющаяся область получила сигнал завершения (например, из-за возникновения ошибки во внешней области или по достижении условия завершения внешней области), тогда все запущенные действия и обработчики внутри данной области терминируются. Scope завершается неуспешно.

В следующих разделах более детально описаны различные типы обработчиков.

3.1.2. Обработка ошибок в области действия

Обработчики ошибок допустимо объявлять в области действия, как указано в листинге 22, для того, чтобы контролировать исключительные ситуации более локально, то есть ближе к месту их возникновения в BPEL-процессе. Графическая нотация обработки ошибок на уровне scope представлена на рис. 31.

```
<scope>
   <faultHandlers>
      <catch faultName="xyz:anExpectedError">...</catch>
      <catchAll><!-- deal with other errors -->
         ...
      </catchAll>
   </faultHandlers>
   <sequence><!-- do work --></sequence>
</scope>
```

Листинг 22. Локальная обработка ошибок в области действия [2]

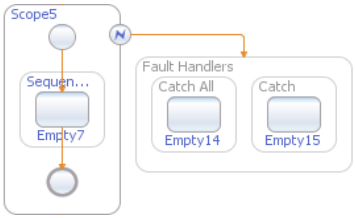


Рис. 31. Локальная обработка ошибок в среде NetBeans

Если исключение выброшено внутри области действия, то данный scope завершается неуспешно. Перед непосредственной остановкой исполнения scope локально определённые обработчики ошибок выполняют свою работу. Обработчики ошибок могут выбрасывать новое исключение или передавать отловленное исключение родительской области действия.

3.1.3. Прерывание исполнения действий

Перед тем, как обработчик ошибок приступит к своей работе, все действия связанной с ним области прерываются. Структурные действия передают сигнал к прерыванию всем действиям, содержащимся в них. Некоторым простейшим действиям (assign, empty, throw, rethrow, exit) позволяется при этом завершить свою работу, остальные же действия прерываются.

Области действия могут самостоятельно контролировать своё поведение при завершении. Такая возможность обуславливается необходимостью выполнения действий по освобождению ресурсов системы, например, по закрытию подключений к базе данных или по отсылке нотификации партнерскому сервису.

Обработчик завершения (terminationHandler) запускается после прерывания выполнения текущего действия в области действий и завершения работы всех обработчиков событий, привязанных к данной области, в результате возникновения исключения. Пример объявления обработчика завершения приведен в листинге 23 и представлен графической нотацией на рис. 32.

```
<scope>

   <terminationHandler>
      <!-- clean up resources in case
           of forced termination
      -->
   </terminationHandler>

   <sequence>
      <!-- do work -->
   </sequence>

</scope>
```

Листинг 23. Обработка завершения scope [2]

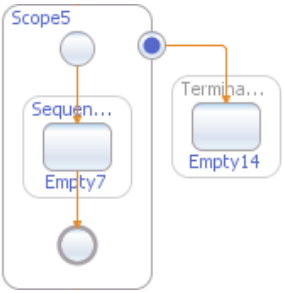


Рис. 32. Обработка завершения scope в среде NetBeans

Обработчики завершения могут содержать любые действия BPEL в том числе compensate и compensateScope. Однако обработчики не могут передавать исключения родительским областям, так как завершение работы было вызвано другим исключением либо выполнением условия остановки действия forEach.

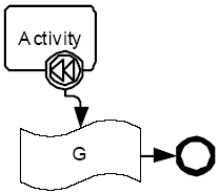
3.1.4. Отмена выполненной работы

Рабочие процессы часто рассчитаны на длительное время выполнения. Процессы BPEL в ходе своего исполнения совершают атомарные действия, которые можно трактовать как ACID транзакции. Таким образом, создается побочный персистентный эффект до момента завершения работы процесса, иными словами, результаты выполнения частей процесса фиксируются. Например, обработка заказа может включать в себя несколько ступеней BPEL-процесса – получение заказа, отправка заявки на требуемый товар на склад, нотификация клиента о выполнении заказа, подтверждение оплаты товара, нотификация склада об отгрузке заказа. При возникновении исключительной ситуации на этапе подтверждения оплаты, необходимо "откатить" уже достигнутые результаты: расформировать заявку на складе. В BPEL существуют специальные языковые конструкции, предназначенные для отмены выполненных шагов процесса. Компенсационный обработчик (compensationHandler) решает данную задачу. Он вызывается после успешного завершения связанной с ним области действия при помощи конструкций compensate или compensateScope, как показано в листинге 24.

```
<scope name="S1">
   <faultHandlers>
      <catchAll><compensateScope target="S2" /></catchAll>
   </faultHandlers>
   <sequence>
      <scope name="S2">
         <compensationHandler>
            <!-- undo work -->
         </compensationHandler>
         <!-- do some work -->
      </scope>
      <!-- do more work -->
      <!-- fault is thrown here; results of S2 must be undone -->
   </sequence>
</scope>
```

Листинг 24. Применение механизма компенсаций [2]

Графическое представление механизма компенсаций в нотации BPMN и в среде NetBeans изображено на рис. 33 и рис. 34.



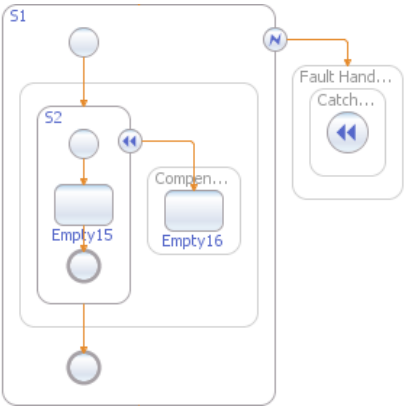


Рис. 33. Обработчик компенсаций в нотации BPMN [3]

Рис. 34. Применение механизма компенсаций в среде NetBeans

Компенсационному обработчику области действия доступна информация о текущем состоянии экземпляра процесса. Состояние успешно завершенной области (scope) сохраняется таким образом, что компенсационный обработчик может продолжить работу с ним в дальнейшем.

Обработчик компенсации вызывается действиями compensate или compensateScope, которые могут находиться внутри обработчиков ошибок, компенсации, завершения (обычно на эти обработчики ссылаются аббревиатурой FCT по начальным буквам их названий в англоязычном варианте – fault, compensation, termination) родительской области действий. Действие compensate запускает компенсационный механизм во всех ещё некомпенсированных дочерних областях в обычном порядке. Действие compensateScope вызывает компенсацию одной указанной области. Если область, связанная с компенсационным обработчиком, содержится в циклической конструкции или в обработчике событий, то тогда количество компенсационных обработчиков соответствует количеству соответствующих им экземпляров scope.

Каждый экземпляр компенсационного обработчика вызывает компенсацию в дочерних областях действий. Набор всех вызываемых таким образом по цепочке компенсационных обработчиков называется группой компенсационных обработчиков данного экземпляра.

Если в обработчике компенсаций выбрасывается исключение, то оно передается области действий, вызвавшей данный обработчик. Перед запуском соответствующего обработчика ошибок завершается вся выполняющаяся работа, в том числе и работа группы компенсационных обработчиков.

3.1.5. Обработка событий

Любая область действия может содержать в себе объявление обработчиков событий. Они служат для обработки входящих сообщений в потоке, параллельном потоку исполнения основного действия области. Обработчики событий рассмотрены ниже в качестве одного из механизмов организации развернутого взаимодействия между сервисами.

3.2. Развернутое взаимодействие веб-сервисов

В предыдущих частях было объяснено, как используется действие recieve для выполнения блокирующего ожидания определенного входящего сообщения. Существуют сценарии рабочих процессов, в которых требуется более сложная схема взаимодействия сервисов. Например, рабочий процесс должен принять сообщение от одного из нескольких веб-сервисов, или ожидается то, что будет получено несколько сообщений от разных сервисов до момента, когда рабочий

процесс сможет перейти к следующему этапу своего исполнения. Иногда сообщения должны быть обработаны параллельно с основным потоком исполнения процесса. Все эти варианты взаимодействия поддерживаются конструкциями языка BPEL. В спецификации BPEL на совокупность данных структур сокращенно ссылаются аббревиатурой IMA (inbound message activities, то есть действия с входящими сообщениями). Далее каждое действие IMA будет детально рассмотрено.

3.2.1. Выборочная обработка событий

Допустим, что существует набор входящих сообщений для процесса, получение любого из них позволяет процессу выполнить соответствующий шаг бизнес-логики. Существует возможность описать поведение системы при отсутствии входящих сообщений в течение заданного периода времени. Действие pick содержит хотя бы один элемент onMessage, а также может содержать сколько угодно элементов onAlarm. Каждый элемент onMessage указывает на операцию веб-сервиса, объявленную в WSDL-интерфейсе, имплементируемом BPEL-процессом. Элемент onMessage содержит указание переменной, предназначенной для хранения входящего сообщения. Каждый элемент onAlarm объявляет точку во времени либо временной промежуток. И onMessage, и onAlarm содержат бизнес-логику, которая выполняется при приеме соответствующего сообщения либо при достижении тайм-аута. Пример использования выборочной обработки событий приведен в листинге 25 и изображен в графической нотации на рис. 35 и рис. 36.

```
<pick>
   <onMessage partnerLink="buyer"
      operation="inputLineItem"
      variable="lineItem">
      <!-- activity to add line item to order -->
   </onMessage>
   <onMessage partnerLink="buyer"
      operation="orderComplete"
      variable="completionDetail">
      <!-- activity to perform order completion -->
   </onMessage>
   <onAlarm>
      <for>'P3DT10H'</for>
      <!-- handle timeout for order completion -->
   </onAlarm>
</pick>
```

Листинг 25. Действие pick [2]

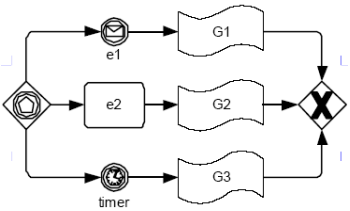


Рис. 35. Действие pick в нотации BPMN [3]

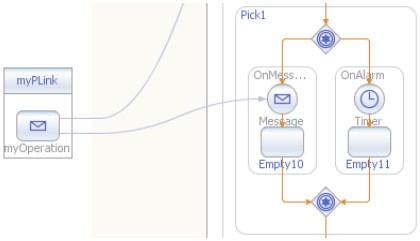


Рис. 36. Действие pick в среде NetBeans

В приведенном примере действие pick содержит два элемента onMessage и один onAlarm. После того, как будет вызвана операция inputLineItem либо операция orderComplete, либо истечет временной интервал в 3 дня и 10 часов, выполнится соответствующая одному из этих событий логика и завершится действие pick.

Аналогично действию recieve действие pick может объявлять атрибут createInstance="yes", что приведет к созданию нового экземпляра BPEL-процесса в случае получения сообщения. Экземпляр процесса может быть создан только тогда, когда выполняется onMessage, но не onAlarm.

3.2.2. Обработка множества событий

В предыдущем разделе было показано, как одно из множества сообщений может быть принято. Теперь обратимся к ситуации, когда процесс может продолжить свое исполнение только после получения нескольких сообщений.

Представим процесс выполнения брокерской сделки, которая заключается только после того, как покупатель (buyer) и продавец (seller) пришли к обоюдному согласию в совершении данной финансовой операции.

Оба участника сделки информируют брокера о своем согласии в участии, затем брокер производит расчет. Процесс брокера ожидает сообщения от обоих партнеров, не зная, какое из них будет получено первым.

```
<flow>
 <links>
  <link name="buyToSettle" />
  <link name="sellToSettle" />
 </links>
 <receive name="receiveBuyerInfo" createInstance="yes" ...>
  <sources>
   <source linkName="buyToSettle" />
  </sources>
  <correlations>
   <correlation set="tradeID" initiate="join" />
  </correlations>
 </receive>
 <receive name="receiveSellerInfo" createInstance="yes" ...>
  <sources>
   <source linkName="sellToSettle" />
  </sources>
  <correlations>
   <correlation set="tradeID" initiate="join" />
  </correlations>
 </receive>
 <invoke name="settleTrade" ...>
  <targets>
    <joinCondition>$buyToSettle and $sellToSettle</joinCondition>
      <target linkName="buyToSettle" />
      <target linkName="sellToSettle" />
     </targets>
  </invoke>
   ...
</flow>
```

Листинг 26. Пример использования множества инициирующих действий [2]

В вышеприведенном примере в листинге 26 два действия recieve содержат атрибуты createInstance="yes". Подобные действия называют ещё инициирующими. В приведенном сценарии отсутствует необходимость в создании двух разных экземпляров процесса. Сообщения и от покупателя, и от продавца должны быть обработаны одним и тем же экземпляром. Именно это достигается использованием двух инициирующих действий. Если оба входящих сообщения относятся к соответствующим действиям recieve и предназначены для одного экземпляра BPEL-процесса, то второе пришедшее сообщение не создаст новый экземпляр процесса. В следующих разделах будет описано, каким образом данные сообщения коррелируют с необходимым экземпляром процесса. Хотя в данном примере присутствуют два инициирующих действия, реально только первое пришедшее повлечет за собой создание

нового экземпляра процесса, а второе сообщение будет принято этим же экземпляром. После того, как оба действия recieve будут выполнены, логика брокерского процесса продолжит свое выполнение действием settleTrade, объединяющим две параллельно исполняемые ветви структуры flow в один поток.

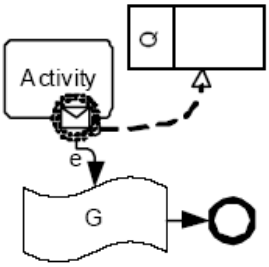
3.2.3. Конкурентная обработка событий

До сих пор рассматривались действия по обработке входящих сообщений, полностью блокирующие BPEL-процесс на время до прихода сообщения либо до достижения таймаута. Существуют такие рабочие процессы, блокирование выполнения которых нежелательно или невозможно по тем или иным причинам.

Представим процесс оформления заказа на покупку, инициируемого сообщением от покупателя. Заказ может быть выполнен без последующих дополнительных взаимодействий: в результате товары будут отправлены, а счет сразу возвращен покупателю. В некоторых случаях покупателю необходимо отслеживать статус обработки заказа, изменять заказ и даже отменять его во время выполнения. Подобные взаимодействия с покупателем не могут быть запланированы только на определенных шагах выполнения процесса, но BPEL-процесс должен иметь возможность обрабатывать подобные запросы параллельно его основному потоку. В языке BPEL такие асинхронные вызовы называются обработкой событий. Пример объявления обработчиков событий приведен в листинге 27 и графически изображен в нотации на рис. 37 и рис. 38.

```
<process name="purchaseOrderProcess" ...>
   ...
   <eventHandlers>
      <onEvent partnerLink="purchasing"
         operation="queryOrderStatus" ...>
         <scope>...</scope>
      </onEvent>
      <onEvent partnerLink="purchasing"
         operation="cancelOrder" ...>
         <scope>...</scope>
      </onEvent>
   </eventHandlers>
   ...
</process>
```

Листинг 27. Обработчики событий [2]



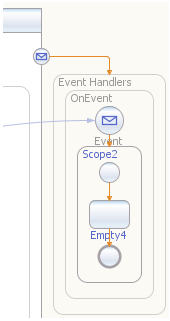


Рис. 37. Обработчик событий в нотации BPMN [3]

Рис. 38. Обработчик событий в среде NetBeans

Обработчики событий ассоциированы со всем процессом или с отдельной областью действий. Обработчики начинают свою работу тогда, когда их область инициализируется, и заканчивают, когда она завершается. Во время работы обработчика может произойти любое количество событий. Все они обрабатываются параллельно основному действию области и параллельно друг другу. Упоминаемые здесь события представляют соответствующие операции веб-сервиса, определенного в WSDL-интерфейсе, имплементируемом BPEL-процессом. События обрабатываются при помощи onEvent элементов. Так же как в действиях pick, в обработчиках событий можно использовать элементы onAlarm, накладывающие ограничения по времени на ожидание входящих сообщений. Однако в обработчиках событий действие, связанное с элементом onAlarm, может выполняться несколько раз. Обработчики не могут создавать новые экземпляры процесса, поэтому события приходят в уже созданный активный экземпляр.

3.2.4. Корреляция сообщений

В предыдущих секциях представлено несколько сценариев, в которых процесс получал более одного входящего сообщения, иными словами, он имплементировал более одной операции веб-сервиса. Некоторые действия, ассоциированные с приемом входящих сообщений,

могут создавать новые экземпляры процесса, другие же, напротив, служат для моделирования ситуаций, когда выполняющийся экземпляр процесса получает дополнительные запросы.

Возникает вопрос, каким образом входящее сообщение "находит" экземпляр процесса, для которого оно предназначено? Возможны stateful имплементации веб-сервисов, в которых целевой экземпляр сервиса идентифицируется параметрами, закодированными в заголовке SOAP сообщения. (Более подробно см. спецификацию WS-Addressing). Данный механизм может быть использован и в BPEL-процессах, но спецификация BPEL не ограничивает решение вопроса корреляции только этим подходом. Язык BPEL поддерживает свой независимый корреляционный механизм, называемый корреляционным набором.

Основной идеей этой корреляционной концепции является тот факт, что большинство сообщений, которыми процессы обмениваются с окружающей средой, уже содержат данные, позволяющие уникально идентифицировать необходимый экземпляр процесса. Например, представим снова процесс заказа товара. Клиент, инициирующий заказ, обычно передает вместе с запросом свой идентификатор клиента. Что если несколько заказов от одного и того же клиента одновременно находятся в процессе выполнения? В этом случае уникальным ключом заказа будет являться идентификатор заказа. Как правило, данная информация уже содержится в структуре передаваемого сообщения. Язык BPEL позволяет определять свойства, представляющие части корреляционной информации. Сервер исполнения BPEL-процессов знает об этих свойствах и об их расположении в теле сообщения благодаря так называемым синонимам свойств (propertyAlias). В итоге, каждое действие по приему входящих сообщений может быть ассоциировано с набором свойств, который позволяет выполнить корреляцию входящего запроса с уникальным экземпляром процесса.

```
<wsdl:definitions ...
   xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
   ...
 <wsdl:message name="sendPOResponse">
    <wsdl:part name="confirmation"
       element="po:sendPurchaseOrderResponse" />
 </wsdl:message>
 <wsdl:message name="queryPORequest">
  <wsdl:part name="query"
             element="po:queryPurchaseOrderStatus" />
 </wsdl:message>
 <wsdl:message name="cancelPORequest">
  <wsdl:part name="cancellation"
             element="po:cancelPurchaseOrder" />
```

```
 </wsdl:message>
   ...

   <vprop:property name="customerID" type="xsd:string" />
   <vprop:property name="orderNumber" type="xsd:int" />

   <vprop:propertyAlias propertyName="tns:customerID"
      messageType="tns:sendPOResponse" part="confirmation">
      <vprop:query>CID</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:orderNumber"
      messageType="tns:sendPOResponse" part="confirmation">
      <vprop:query>Order</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:customerID"
      messageType="tns:queryPORequest" part="query">
      <vprop:query>CID</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:orderNumber"
      messageType="tns:queryPORequest" part="query">
      <vprop:query>Order</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:customerID"
      messageType="tns:cancelPORequest" part="cancellation">
      <vprop:query>CID</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:orderNumber"
      messageType="tns:cancelPORequest" part="cancellation">
      <vprop:query>Order</vprop:query>
   </vprop:propertyAlias>
   ...

</wsdl:definitions>
```

Листинг 28. Определения свойств корреляции и их псевдонимов [2]

Определение WSDL-сообщений, содержащих корреляционную информацию, показано в вышеприведенном примере в листинге 28. Два корреляционных свойства используются для того, чтобы уникальным образом идентифицировать экземпляр процесса для конкретного заказа покупки товара. Для каждого WSDL-сообщения и каждого свойства объявляется свой псевдоним, который определяет местоположение свойства внутри сообщения, как указано в листинге 28.

```
<process name="purchaseOrderProcess" ...>
   <correlationSets>
      <correlationSet name="PurchaseOrder"
         properties="cor:customerID cor:orderNumber" />
   </correlationSets>
```

```
   ...
   <eventHandlers>
      <onEvent partnerLink="purchasing"
         operation="queryPurchaseOrderStatus" ...>
         <correlations>
            <correlation set="PurchaseOrder" initiate="no" />
         </correlations>
         <scope>...</scope>
      </onEvent>
      <onEvent partnerLink="purchasing"
         operation="cancelPurchaseOrder" ...>
         <correlations>
            <correlation set="PurchaseOrder" initiate="no" />
         </correlations>
         <scope>...</scope>
      </onEvent>
   </eventHandlers>
   ...
   <sequence>
      <receive partnerLink="purchasing"
         operation="sendPurchaseOrder" ...
         createInstance="yes">
      </receive>
      ...
      <reply partnerLink="purchasing"
         operation="sendPurchaseOrder" ...>
         <correlations>
            <correlation set="PurchaseOrder" initiate="yes" />
         </correlations>
      </reply>
      ...
      <!-- process the purchase order -->
      ...
   </sequence>
</process>
```

Листинг 29. Корреляционные наборы [2]

В приведенном в листинге 29 примере процесса его экземпляр создается при получении нового заказа. Счет по текущему заказу возвращается клиенту действием reply. Ответное сообщение содержит корреляционную информацию, которая должна присутствовать в последующих обращениях к данному экземпляру процесса. В связи с этим каждый обработчик событий ссылается на тот же корреляционный набор. Последний инициализируется, когда отсылается сообщение о приеме заказа. С этого момента корреляционный набор становится неизменяемым и идентифицирует данный экземпляр процесса. Обработчики событий для проверки статуса заказа и его отмены устанавливают атрибут initiate="no". Когда соответствующие WSDL-операции вызываются,

корреляционные свойства customerId, orderNumber должны соответствовать тем значениям, которые они получили при выполнении действия reply, иначе запрос не сможет коррелировать ни с одним экземпляром процесса.

3.2.5. Конкурентный обмен сообщениями

Для реализации WSDL-операций типа запрос-ответ каждое действие по обработке входящих сообщений (recieve, onMessage, onEvent) может иметь ассоциированный с ним элемент reply. Если процесс имеет несколько действий recieve и reply, указывающих на одну и ту же партнерскую связь и WSDL-операцию, то возникает неопределенность в соответствии элементов recieve действиям reply. В этом случае необходимо указать взаимосвязь сообщений, используя атрибут messageExchange для того, чтобы разрешить неоднозначность, как указано в листинге 30.

```
<process ...>
   ...
   <messageExchanges>
      <messageExchange name="receiveBuyerInformation" />
      <messageExchange name="receiveSellerInformation" />
   </messageExchanges>
   ...
   <flow>
      ...
      <receive messageExchange="receiveBuyerInformation"
         partnerLink="tradingPartner"
         operation="tradingPartnerInfo" ... />
      <receive messageExchange="receiveSellerInformation"
         partnerLink="tradingPartner"
         operation="tradingPartnerInfo" ... />
      ...
      <reply messageExchange="receiveBuyerInformation"
         partnerLink="tradingPartner"
         operation="tradingPartnerInfo" ... />
      <reply messageExchange="receiveSellerInformation"
         partnerLink="tradingPartner"
         operation="tradingPartnerInfo" ... />
   </flow>
</process>
```

Листинг 30. Механизм явного объявления взаимосвязей в процессе обмена сообщениями [2]

В приведенном примере две пары recieve-reply указывают на одну и ту же партнерскую связь и операцию веб-сервиса. Для разрешения неопределенности во взаимосвязях между recieve и reply определены два элемента messageExchange, ссылки на которые даны в соответствующих парах recieve-reply.

Что произойдет, если в качестве реакции на входящее сообщение операции типа request-response не будет отправлено ответное сообщение? Допустим, некоторая область действий объявляет элемент messageExchange и/или партнерскую связь. Если действие по обработке входящего сообщения (IMA, receive, pick/onMessage, eventHandlers/onEvent), ссылающееся на данный элемент messageExchange или на партнерскую связь, выполнится, а соответствующее ему парное действие reply выполнено не будет до завершения области действий, то в таком случае действие IMA будет считаться как бы "осиротевшим". В случае обнаружения подобных IMA выбрасывается стандартное исключение bpel:missingReply.

3.3. Дополнения к параллельным вычислениям

Ранее описаны циклические конструкции языка BPEL: действия while, repeatUntil, forEach. В дополнение к этим последовательным циклам существует параллельная версия цикла forEach – вместо последовательного выполнения каждой итерации цикла выполнение всех итераций начинается одновременно в разных потоках. Указанный способ параллельных вычислений является одним из трех, доступных в BPEL, другие два – это действие flow и обработчики событий.

Параллельная версия цикла forEach используется в сценариях, когда необходимо обработать независимые друг от друга наборы данных или для одновременного независимого обращения к различным партнерским сервисам. Основное отличие данной версии forEach от действия flow заключается в том, что в forEach неизвестно заранее точное количество параллельных ветвей процесса, так как оно определяется не в режиме моделирования, а на этапе исполнения. В действии forEach определяется переменная, являющаяся счетчиком, используемым для итерации по всем параллельным ветвям цикла. Значение данной переменной ограничено верхним и нижним пределами. Она также может быть использована для доступа по индексу к элементам, объявленным в XML-схеме с атрибутами minOccurs и maxOccurs, значения которых больше единицы.

Действие forEach также позволяет задать условие завершения цикла. Его можно использовать тогда, когда не требуется, чтобы все параллельно исполняемые ветви цикла завершили свою работу. Например, когда в forEach были отправлены сообщения нескольким партнерским сервисам, и большинство из них вернули ответные сообщения с необходимой информацией.

В следующем примере в листинге 32 продемонстрировано применение действия forEach. Пример имеет следующую структуру:

- получить список партнерских сервисов (EPR);
- проинициализировать список котировок;
- для EPR каждого партнера произвести в отдельном потоке:
    - присвоить EPR партнерской связи, локальной к области действий forEach;
    - отправить запрос котировки у партнерского сервиса;
    - получить значение котировки;
- завершить выполнения действия forEach, когда 50% партнеров пришлют ответные сообщения;
- обработать полученные котировки.

В примере используется файл XSL-трансформации "initQuotes.xsl" (листинг 31) для того, чтобы инициализировать выходную переменную для списка котировок: для каждого партнерского partnerEPR создается пустой элемент списка котировок (quotes).

```
<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
  xmlns:bo="http://example.com/bo">
  <xsl:template match="bo:partnerEPRs">
        <foo:responses>
        <xsl:for-each select="sref:service-ref">
        <bo:quote />
        </xsl:for-each>
        </foo:responses>
  </xsl:template>
</xsl:transform>
```

Листинг 31. Файл XSL-трансформации для создания списка котировок [2]

Код BPEL-процесса, структура которого описана выше, выглядит следующим образом.

```
<invoke name="retrieveBusinessPartners"
  partnerLink="localDirectory"
  operation="retrieveBusinessPartners"
  inputVariable="filterCriteria"
  outputVariable="partnerEPRs" />

<assign><!-- initialize list of quotes -->
  <copy>
        <from>
        bpel:doXslTransform("initQuotes.xsl",$partnerEPRs)
        </from>
        <to>$quotes</to>
  </copy>
</assign>

<forEach parallel="yes" countername="n">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>
        count($partnerEPRs/sref:service-ref)
  </finalCounterValue>
  <completionCondition>
        <branches>ceiling(0.5*count($partnerEPRs/sref:service-
ref))</branches>
  </completionCondition>
  <scope>
        <partnerLinks>
        <partnerLink name="address" partnerLinkType="..."
        partnerRole="..." myRole="..." />
        </partnerLinks>

        <sequence>
        <assign><!-- get one business partner's EPR -->
        <copy>
               <from>$partnerEPRs[$n]</from>
               <to partnerLink="address" />
        </copy>
        </assign>

        <invoke name="requestQuote"
        partnerLink="address"
        operation="requestQuote"
        inputVariable="quoteRequest" />

        <receive name="receiveQuote"
        partnerLink="address"
        operation="receiveQuote"
        variable="quote" />

        <assign>
        <copy>
               <from variable="quote" />
```

```
                <to>$quotes[$n]</to>
        </copy>
        </assign>
        </sequence>

  </scope>
</forEach>

<!-- ... work with the list of received quotes ... -->
```

Листинг 32. Пример использования параллельной версии цикла forEach [2]

3.4. Отложенное исполнение

Встречаются ситуации, когда выполнение BPEL-процесса необходимо приостановить на определенное время либо до какого-то момента времени. Для реализации такой функциональности в языке BPEL предусмотрено действие wait, оно служит для приостановки выполнения процесса. Пример его использования приведен в листинге 33, а графическая нотация представлена на рис. 39 и рис. 40. Следует отметить, что при использовании wait потоки, параллельные текущему потоку, не будут приостановлены.

```
<wait><!-- wait for three days and ten hours to go by ... -->
   <for>'P3DT10H'</for>
</wait>
```

Листинг 33. Действие wait

Рис. 39. Действие wait в нотации BPMN [3]

Рис. 40. Действие wait в среде NetBeans

Действие wait может принимать в качестве аргумента XPath-выражения двух типов, которые задают:

- временной интервал приостановки исполнения;
- дату и время возобновления исполнения.

Если задан отрицательный или нулевой интервал либо время возобновления исполнения уже прошло, то действие wait завершается сразу после старта.

3.5. Немедленное завершение процесса

Когда в BPEL-процессе возникают исключительные ситуации, он может обработать их, используя механизмы обработки ошибок, обработки завершения и компенсационный механизм, описанные ранее. В случае возникновения серьезных системных ошибок не всегда целесообразно применение данных механизмов. Действие exit может быть использовано, чтобы немедленно завершить все запущенные действия процесса на всех параллельных ветвях исполнения без вызова обработчиков ошибок, завершения и компенсации. Пример его использования приведен в листинге 34, а графическая нотация изображена на рис. 41 и рис. 42.

```
<if>
   <condition>...</condition><!-- the good case -->
   ...
   <elseif>
      <condition>...</condition><!-- handle expected error -->
      ...
   </elseif>
   ...
   <else><!-- unexpected error – unable to handle ... -->
      <exit />
   </else>
</if>
```

Листинг 34. Действие exit [2]

Рис. 41. Действие exit в нотации BPMN [3]

Рис. 42. Действие exit в среде NetBeans

Следует помнить, что действие exit также прерывает все открытые сессии обмена сообщениями с партнерами процесса, поэтому может возникнуть ситуация, когда партнерский сервис будет ожидать сообщения от BPEL-процесса, которое никогда не будет отправлено.

3.6. Пустое действие

Действие empty используется для того, чтобы явно сигнализировать о том, что никакое реальное действие выполняться не будет. Существует несколько случаев, когда используется данное действие:

- в обработчиках ошибок, когда необходимо подчеркнуть, что исключение отлавливается, но никаких действий, связанных с ним, не производится;
- для объявления точек синхронизации в потоке;
- в качестве заглушек действиям, которые будут добавлены позже в ходе разработки.

Пример использования действия empty приведен в листинге 35. Графическая нотация данного элемента изображена на рис. 43 и рис. 44.

```
<empty>
   <targets>
      <target linkName="left-branch" />
      <target linkName="right-branch" />
   </targets>
</empty>
```

Листинг 35. Действие empty [2]

Рис. 43. Действие empty в нотации BPMN [3]

Рис. 44. Действие empty в среде NetBeans

3.7. Валидация данных

Процесс BPEL получает данные от партнеров через действия получения входных сообщений. Действие assign, описанное ранее, предоставляет простейший способ обработки данных. Может произойти так, что результат данного действия, произведенного над переменной BPEL, не будет валидным с точки зрения WSDL-сообщения, XSD-типа или XSD-элемента, связанного с данной переменной при её декларации. Переменные сложных типов могут инициализироваться за несколько шагов, в связи с этим иногда возникает необходимость, чтобы промежуточные результаты такой инициализации были также валидными.

Для того, чтобы убедиться в соответствии содержимого переменной типу, указанному при декларации, можно воспользоваться действием validate или атрибутом validate действия assign, предназначенными для валидации данных.

```
<scope>
   <faultHandlers>
      <catch faultName="bpel:invalidVariables">
         <reply name="invalidPurchaseOrder" ... />
      </catch>
   </faultHandlers>
   <sequence>
      <receive name="receivePurchaseOrder"
               variable="purchaseOrder" ... />
      ...
      <validate name="validatePurchaseOrder"
                variables="purchaseOrder" />
      <reply name="acknowledgeReceipt" ... />
      ...
   </sequence>
</scope>
```

Листинг 36. Действие validate [2]

Действие validate в листинге 36 проверяет соответствие каждой переменной из переданного списка её XML-определению, данному в XSD-схеме или WSDL-документе. Если атрибут validate в действии assign имеет значение "yes", то тогда проверка соответствия схеме производится для всех измененных переменных. В случае, если переменная объявлена как WSDL-сообщение, то все части сообщения будут проверены. Если в ходе проверки выявляются ошибки, то выбрасывается стандартное исключение bpel:invalidVariables.

Явная валидация переменных в ходе исполнения процесса требует дополнительных ресурсов, что неприемлемо для высоконагруженных

систем. В связи с этим, как правило, системы, отвечающие за исполнение BPEL-процессов, позволяют включать и отключать валидацию с помощью соответствующей настройки среды без изменения исходного кода процесса.

3.8. Конкурентная обработка данных

Ранее описано несколько способов создания многопоточных процессов в BPEL. Модель процесса на языке BPEL может содержать:

- заранее определенное количество параллельных действий, созданных с помощью действия flow;
- переменное количество параллельных ветвей цикла forEach;
- любое количество экземпляров обработчиков событий.

Ни для кого не секрет, что параллельные вычисления требуют особой осторожности, если используют в своей работе одни и те же глобальные данные.

Область действий scope в языке BPEL обладает механизмом, позволяющим контролировать доступ к глобальным данным процесса. Допустим, существует процесс, состоящий из нескольких областей действий, действия каждой из которых считывают или устанавливают значения общих глобальных переменных. Если атрибут isolated всех этих областей принимает значение "yes", то доступ к общим переменным становится защищенным с точки зрения конкурентной обработки данных. Иными словами, если действия в конкурентных областях получают доступ к общим переменным, то эти операции доступа синхронизируются так же как, если бы все действия одной области выполнялись до выполнения действий в других областях. Пример использования данного механизма приведен в листинге 37.

```
<process ...>
   <variables>
      <variable name="global" element="..." />
   </variables>
   <flow>
      <scope name="S1" isolated="yes">
         <sequence>
            ...
            <invoke ... outputVariable="global" />
            ...
         </sequence>
      </scope>
      <scope name="S2" isolated="yes">
```

```
         <sequence>
            ...
            <assign>
               <copy>
                  <from>...</from>
                  <to variable="global" />
               </copy>
            </assign>
            ...
         </sequence>
      </scope>
   </flow>
</process>
```

Листинг 37. Изолированные области с доступом к глобальной переменной [2]

Подобная семантика изоляции применима для доступа к переменным через свойства наборов корреляций и при использовании EPR в операциях доступа к партнерским связям.

Свойство изоляции распространяется на обработчики событий, ошибок и завершения данной области действий. Компенсационные обработчики изолированной области также принимают свойство изоляции.

Если изолированная область действий содержит дочерние области, то атрибут isolated последних должен принимать значение "no" (если атрибут не указан, то его значение по умолчанию является "no"). Доступ к общим переменным из таких дочерних областей контролируется их изолированной родительской областью.

3.9. Динамическое разрешение партнерских связей

Перед тем как партнерская связь может быть использована для вызова операций партнерского веб-сервиса, она должна быть однозначно ассоциирована с конкретной ссылкой на точку доступа к веб-сервису. Указанная ассоциация может быть установлена либо в момент разворачивания BPEL-процесса на сервере (описание данного способа выходит за рамки спецификации BPEL), либо динамически в ходе исполнения.

Для присвоения ссылки точки доступа (EPR) партнерской связи используется вариация действия assign. Ссылка должна быть определена

согласно спецификации WS-Addressing и должна быть заключена в контейнерный элемент sref:service-ref, как показано в листинге 38.

```
<sref:service-ref
   reference-scheme="http://www.w3.org/2005/08/addressing">
   <wsa:EndpointReference
      xmlns:wsa="http://www.w3.org/2005/08/addressing">
      ...
   </wsa:EndpointReference>
</sref:service-ref>
```

Листинг 38. Объявление ссылки точки доступа к Веб-сервису [2]

Рассмотрим пример из листинга 39. Процесс получает ссылку точки доступа, которая затем используется, чтобы впоследствии вызвать по ней веб-сервис.

```
<process name="purchaseOrderProcess" ...>
   <partnerLinks>
      <partnerLink name="myCustomer"
         partnerLinkType="lns:purchaseOrderLT"
         myRole="purchaseOrderService" />
      partnerRole="customer"/>
   </partnerLinks>
   ...
   <receive partnerLink="myCustomer"
      operation="sendPurchaseOrder"
      variable="purchaseOrder" />
   ...
   <assign>
      <copy>
         <from>$purchaseOrder/callback</from>
         <to partnerLink="myCustomer" />
      </copy>
   </assign>
   ...
   <invoke partnerLink="myCustomer"
      operation="sendInvoice"
      variable="invoice" />
   ...
</process>
```

Листинг 39. Присвоение ссылки точки доступа партнерской связи [2]

В приведенном примере BPEL-процесс объявляет одну одностороннюю операцию для регистрации заказа на покупку товара. Заказ содержит ссылку на точку доступа к веб-сервису (обернутую в контейнерный элемент sref:service-ref), которая указывает на другую

одностороннюю операцию, предоставляемую заказчиком. После того, как ссылка точки доступа будет присвоена партнерской связи, ассоциированной с веб-сервисом заказчика, может быть вызвана операция отсылки счета (sendInvoice).

Рассмотрим случай, когда вызывающее приложение заказчика реализовано как BPEL-процесс. Этот процесс должен послать запрос на регистрацию заказа, включив в передаваемое сообщение ссылку точки доступа, по которой будет отправлен счет. Данный процесс также должен имплементировать операцию получения счета.

```
<process name="customerProcess" ...>
   <partnerLinks>
      <partnerLink name="myPurchaseOrderService"
         partnerLinkType="lns:purchaseOrderLT"
         partnerRole="purchaseOrderService" />
      myRole="customer"/>
   </partnerLinks>
   ...
   <assign>
      <copy>
         <from partnerLink="myPurchaseOrderService"
            endpointReference="myRole" />
         <to>$purchaseOrder/callback</to>
      </copy>
   </assign>
   ...
   <invoke partnerLink="myPurchaseOrderService"
      operation="sendPurchaseOrder"
      variable="purchaseOrder" />
   ...
   <receive partnerLink="myPurchaseOrderService"
      operation="sendInvoice"
      variable="invoice" />
   ...
</process>
```

Листинг 40. Присвоение партнерской связи ссылке точке доступа [2]

Пример, приведенный выше в листинге 40, иллюстрирует реализацию процесса заказчика. Ссылка точки доступа, ассоциированная с сервисом заказчика, копируется в элемент, содержащийся в сообщении заказа. Далее она передается в сервис регистрации заказа, который в своё время использует её для отсылки счета.

Следует отметить, что в данном примере необходимо соблюсти следующее условие: оформленный счет должен высылаться тому экземпляру процесса заказчика, который отправил запрос на регистрацию

заказа. Указанное условие может быть достигнуто либо использованием механизма корреляционных наборов, имплементированного в языке BPEL, либо с помощью подхода, описанного в спецификации WS-Addressing.

4. Расширенное использование языка

4.1. Расширение языка

Язык BPEL расширяем. Любая имплементация языка может поддерживать языки запросов и выражений в дополнение к обязательному языку XPath 1.0. Более того, в BPEL-процессе допустимо использование дополнительных XML-элементов и атрибутов из пространств имен, отличных от пространства имен BPEL.

4.1.1. Языки запросов и выражений

Для того, чтобы осуществлять доступ к структурам данных и обрабатывать их во многих элементах BPEL, используются запросы и выражения, которые по умолчанию пишутся на языке XPath. Поддержка XPath 1.0 гарантируется спецификацией языка BPEL.

В следующем примере в листинге 41 XPath-выражение "$counter < 42" используется в качестве условия в цикле while.

```
<while>
   <condition>$counter &lt; 42</condition>
   ...
</while>
```

Листинг 41. XPath-выражение в качестве условия цикла [2]

Имплементация BPEL может поддерживать дополнительные языки запросов и выражений. В качестве примера рассмотрим булево выражение на языке Java. Для упрощения примера будем использовать "http://www.example.com/java" как значение атрибута expressionLanguage. При этом предполагаем, что значение BPEL-переменной counter представлено Java-переменной с таким же именем.

```
<while>
   <condition expressionLanguage="http://www.example.com/java">
      counter &lt; 42
   </condition>
   ...
</while>
```

Листинг 42. Условие цикла в виде Java-выражения [2]

В приведенном примере в листинге 42 следует обратить внимание на то, что элемент, содержащий выражение, дополнен атрибутом expressionLanguage, и переменная counter используется в выражении так, как она используется в языке Java, то есть без предстоящего символа '$'.

4.1.2. Дополнительные элементы и атрибуты

Почти все элементы BPEL-процесса могут быть расширены дополнительными элементами и атрибутами из произвольного пространства имен. В следующем примере в листинге 43 действие invoke содержит дополнительный дочерний элемент, используемый для представления модели процесса в специализированном редакторе BPEL.

```
<process name="purchaseOrderProcess" ...
   xmlns:tool="http://example.com/bpel/editorElements">
   ...
   <invoke partnerLink="shipping"
      operation="requestShipping"
      inputVariable="shippingRequest"
      outputVariable="shippingInfo">
      <documentation>decide on shipper</documentation>
      <tool:myIcon>shipping/requestShipping.gif</tool:myIcon>
   </invoke>
   ...
</process>
```

Листинг 43. Расширение действия invoke дополнительным дочерним неисполняемым элементом [2]

Для того, чтобы избежать неоднозначности с точки зрения XML схемы документа, точки для расширения языка должны быть обернуты в специальные элементы: extensionActivity и extensionAssignOperation. Еще одним подобным элементом-оберткой является элемент literal. Он служит

для размещения в нем XML-данных в действии присвоения значения переменной.

Следующий пример (листинг 44) иллюстрирует моделирование нестандартного взаимодействия пользователей, реализованное с помощью дополнительных элементов с использованием обертки extensionActivity.

```
<process name="purchaseOrderProcess" ...
   xmlns:user="http://example.com/bpel/userInteractions">
   ...
   <if>
      <condition>$amount &gt; 1000000</condition>
      <extensionActivity>
         <user:userInteraction type="user:approval">
            <user:userResolution role="manager" />
         </user:userInteraction>
      </extensionActivity>
   </if>
   ...
</process>
```

Листинг 44. Расширение BPEL-процесса исполняемым элементом [2]

Язык BPEL предоставляет возможность явно разграничивать пространства имен элементов с семантикой времени исполнения и времени моделирования. Данное разграничение необходимо для того, чтобы указать имплементации BPEL те элементы, которые должны быть ею поняты и обработаны соответствующим образом. В представленном ниже примере используются пространства имен, определенные в двух предыдущих примерах.

Элементы некоторых пространств имен используются только в инструменте моделирования BPEL-процессов и не содержат семантики времени исполнения (элементы с префиксом tool), поэтому они могут быть объявлены с атрибутом mustUnderstand="no" и игнорированы в ходе исполнения процесса. Другие элементы из пространства с префиксом user представляют собой интегральную часть логики процесса. Их пространство объявляется с атрибутом mustUnderstand="yes". Следовательно, если имплементация BPEL не знает, каким образом интерпретировать данные элементы, то такой процесс не должен запускаться. Пример использования данных атрибутов приведен в листинге 45.

```
<process ...
   xmlns:tool="http://example.com/bpel/editorElements"
   xmlns:user="http://example.com/bpel/userInteractions">
```

```
   <extensions>
      <extension
         namespace="http://example.com/bpel/editorElements"
         mustUnderstand="no" />
      <extension
         namespace="http://example.com/bpel/userInteractions"
         mustUnderstand="yes" />
   </extensions>
   ...
</process>
```

Листинг 45. Объявление расширений языка [2]

4.2. Абстрактные процессы

Абстрактные процессы описывают поведение процесса частично, не раскрывая детально всех нюансов его исполнения. Абстрактный процесс может быть имплементирован набором исполняемых процессов. Элементы и атрибуты исполняемого процесса могут быть скрыты в абстрактном процессе: их либо опускают вовсе, либо заменяют заглушками. Языковые конструкции как абстрактного, так и исполняемого процессов используют одинаковую семантику.

Использование абстрактных процессов предпочтительно в следующих случаях:

- абстракция – применяют тогда, когда необходимо показать только конкретные аспекты исполняемого процесса, например, внешние взаимодействия с использованием определенных партнерских связей;
- улучшение – используют как начальную точку разработки исполняемого процесса;
- разработка протокола взаимодействия – проводят для разрешения проблем многоэтапного протокола взаимодействия для двух и более процессов.

Примеры создания абстрактных процессов приведены в [2].

5. Технологический процесс разработки

В данном разделе рассмотрены главные этапы технологического процесса разработки приложений, использующих BPEL-процессы в

качестве основы исполнительной логики. Сначала дано краткое описание структуры технологического процесса, затем приведено подробное описание практической реализации этапов разработки.

5.1. Структура технологического процесса

Технологический процесс разработки сервис-ориентированных средств с использованием языка BPEL разделяется на следующие этапы.

- **Выбор и установка программной платформы.** Это зачастую один из наиболее трудоемких этапов, однако, он делается единожды и, как правило, его результаты распространяются на все последующие проекты в компании. Сложность задачи выбора оптимального варианта программной платформы заключается в её многокритериальности. Оцениваться могут кроме быстродействия в различных конфигурациях ещё лицензионные ограничения, цена решения, поддержка международных технических стандартов, удобство разработки, удобство развертывания и мониторинга, сложность модификации и поддержки программных продуктов, открытость исходных кодов инструментов и т.д.
- **Создание архитектуры системы.** Основными задачами на данном этапе являются анализ требований, составление вариантов использования, эффективное распределение функциональности по компонентам системы. Необходимо выделять переиспользуемую и редко меняющуюся функциональность в виде сервисов в терминах SOA. Более динамично развивающиеся части функциональности следует выделять в виде бизнес-процессов. Более подробная информация на данную тему изложена в [4].
- **Построение аналитической модели интеграции.** Аналитическая модель позволяет выполнить математическую оценку вероятностных характеристик сервисного обслуживания и выбрать те модели интеграции сервис-ориентированных средств, которые удовлетворяют объективным требованиям.
- **Кодирование веб-сервисов и бизнес-процессов.** Веб-сервисы и бизнес-процессы могут разрабатываться независимо после фиксации интерфейсов взаимодействия.

- **Интеграция компонентов.** Производится, как правило, специалистом по интеграции или администратором системы.
- **Тестирование комплекса.** В данный этап входит функциональное, нагрузочное и другие виды тестирования.

5.2. Выбор программной платформы

Перед непосредственной разработкой бизнес-логики приложения необходимо провести подготовительный этап, заключающийся в инсталляции программного обеспечения, отвечающего за разработку и исполнение BPEL-процессов.

5.2.1. Среды разработки и исполнения BPEL

Существует ряд средств разработки, поддерживающих графический интерфейс редактирования графов бизнес-процессов, что существенно облегчает труд программистов и бизнес-аналитиков и сокращает издержки на разработку приложений с использованием технологии управления бизнес-процессами.

К подобным системам относятся, например, следующие:

- NetBeans IDE;
- Eclipse IDE BPEL;
- Oracle Jdeveloper.

Вышеприведенные примеры диаграмм бизнес-процессов были взяты из редактора диаграмм BPEL среды разработки NetBeans 6.5. Так как технологический процесс во многом определяется средой разработки, то рассмотрим только данную среду, а исследование остальных продуктов вынесем за пределы данного пособия.

Среда NetBeans является бесплатной программной платформой. Она доступна на ресурсе: http://netbeans.org/downloads/6.5.1/.

В сборку системы вместе со средой разработки входит бесплатно распространяемый сервер приложений GlassFish V2.1, являющийся в совокупности с входящим в него компонентом BpelEngine средой исполнения BPEL-процессов. Указанный сервер приложений, являясь независимым проектом разработки, может быть установлен отдельно с адреса: http://glassfish.java.net/downloads/v2.1.1-final.html. Однако после инсталляции необходимо установить дополнение, поддерживающее исполнение BPEL (<install-dir>glassfish/updatecenter/bin/updatetool->Composite Applications).

Мастер установки NetBeans 6.5.1 (рис. 45) поочередно проинсталлирует среды разработки и исполнения при запуске. Для работы с сервис-ориентированными средствами следует сконфигурировать установщик следующими обязательными компонентами:

- SOA;
- Glassfish V2.1;
- OpenESB v2;
- Java Web and EE.

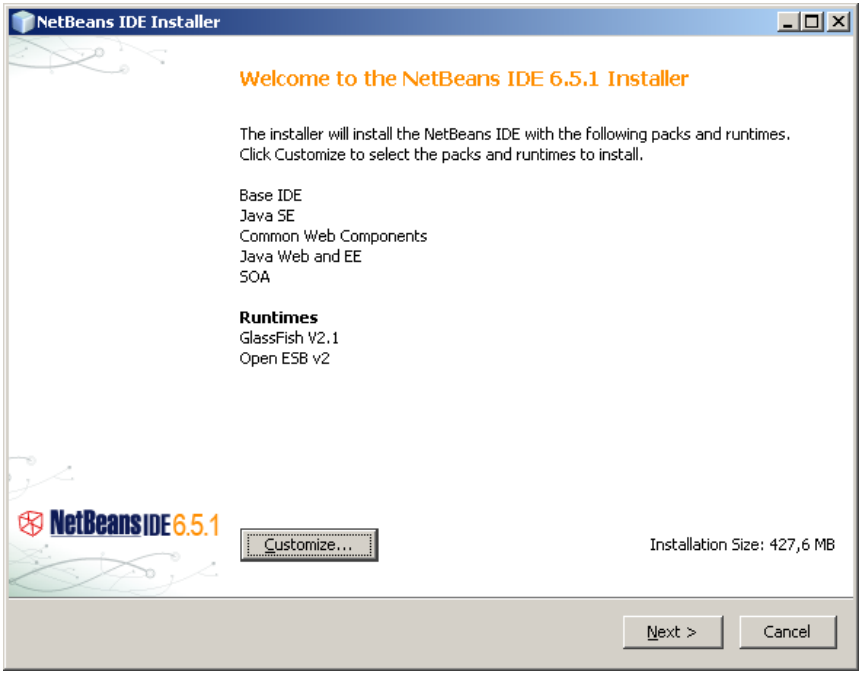


Рис. 45. Окно мастера установки сред разработки и исполнения BPEL-процессов с перечислением требуемых компонентов

После завершения процесса установки на сервере приложений будет развернут домен "domain1", используемый в качестве контейнера для исполнения BPEL-процессов, и среда разработки NetBeans.

5.3. Высокоуровневое описание логики бизнес-процесса

Рассмотрим этап разработки бизнес-процесса на примере создания BPEL-процесса, использующего в своей работе партнерский веб-сервис.

В рамках технологической цепочки разработки и тестирования сервис-ориентированных средств бизнес-процесс и партнерские веб-сервисы могут разрабатываться независимо после определения интерфейсов взаимодействия.

Следует учитывать, что на момент начала разработки интерфейсы могут быть не вполне уточнены, однако данное обстоятельство не препятствует независимости разработки, но накладывает дополнительные накладные расходы для их интеграции и обновления.

В качестве примера рассмотрим простой бизнес-процесс, осуществляющий контроль пользователей при входе в информационную систему.

Основу бизнес-логики рассматриваемого процесса составляют две функции:

- проверки имени пользователя;
- формирования сообщения приветствия.

Данные функции будут распределены между партнерским веб-сервисом и BPEL-процессом. Кроме того, на BPEL-процесс ложится функциональность по взаимодействию с веб-сервисом, называемая также оркестровкой.

Приведем представление бизнес-логики в виде SDL-диаграммы на рис. 46 для того, чтобы читатель впоследствии мог сопоставить её с реальной BPEL-диаграммой исполняемого процесса.

Язык SDL (Specification and Description Language) предназначен для спецификации и описания поведения реактивных распределенных систем. Он широко распространен в сфере телекоммуникационных информационных систем, к которой можно отнести рассматриваемый бизнес-процесс.

Детальную информацию о данном графическом представлении можно найти в [5].

Бизнес-процесс активизируется во время приёма входящего сообщения от пользователя, представляющего собой запрос.

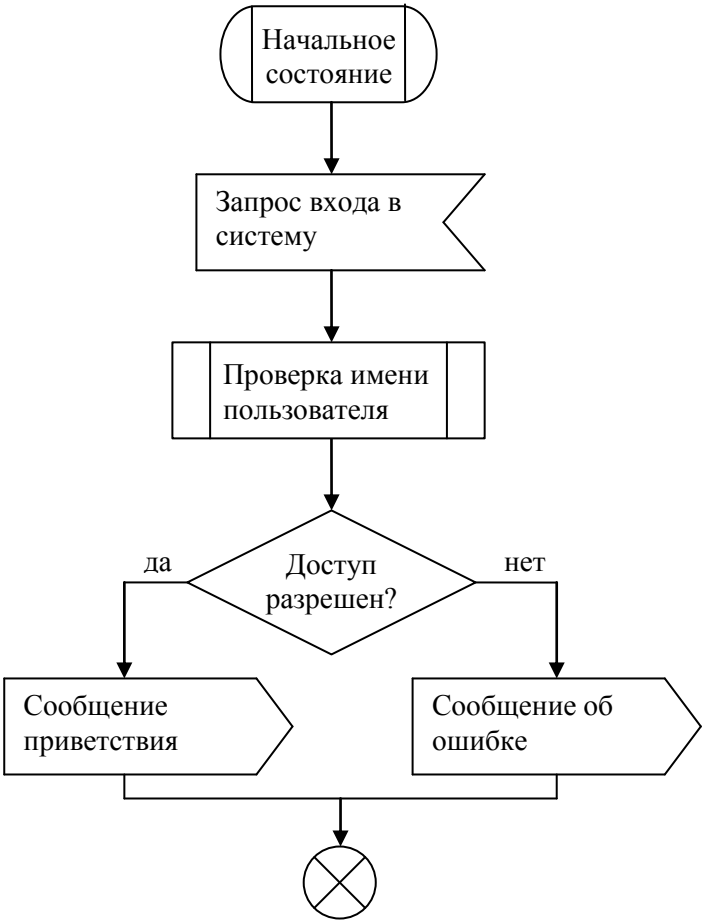


Рис. 46. SDL-диаграмма бизнес-процесса

5.4. Формирование аналитической модели интеграции сервис-ориентированных средств

Расширение модели SOA проводится на уровне описания деятельности. В системе стандартов, соответствующих концепции SOA, на этом уровне используется язык BPEL.

Согласно [6] деятельность рассматривается как самостоятельный элемент поведения приложения, которая может включать другие деятельности или отдельные действия. Под элементарной единицей спецификации поведения понимается действие. В модели деятельности описывается последовательность действий и условий их выполнения на основе определения потока управления, а также поток объектов, необходимых для осуществления отдельных действий или соответствующих результатам их реализации. В русле объектно-ориентированного анализа информационных систем модель изображается в форме диаграммы деятельности (activity diagram).

При сопоставлении спецификаций языка BPEL и языка унифицированного моделирования UML 2.0 (Unified Modeling Language) выясняется, что на разработку каждого из них существенное влияние оказала нотация моделирования бизнес-процессов BPMN (Business Process Modeling Notation). В связи с этим в качестве основы для расширения выбирается система элементов моделирования деятельности языка UML 2.0, тем более, что расширяемость позиционируется в качестве одного из его ключевых свойств.

В концептуальную основу моделирования интеграции сервис-ориентированных средств включаются принципы определения деятельности, действия, узлов и дуг деятельности, семантики деятельности, семантики действия, потока управления, потока объектов, специальных регионов, специальных действий.

Поток управления рассматривается как прототип координации сервис-ориентированных средств, выполняющих действия деятельности в среде сервис-ориентированной архитектуры. Он образуется из узлов, являющихся абстрактными узлами деятельности и предназначающихся для координации потоков. Узлами управления являются начальный узел, узел решения, узел слияния, узел разделения, узел соединения, узел финала деятельности и потока.

Поток объектов формируется с помощью узлов, среди которых узел объекта, узел центрального буфера и хранилища данных, узел входных контактов объекта, узел выходных контактов объекта, узел параметра деятельности.

Группа элементов моделирования, называемая специальные регионы, предназначается для группировки действий, относящихся к одной деятельности и имеющих некоторую общую характеристику. В такую группу включаются элементы разбиения деятельности, элементы региона прерываемой деятельности, элементы обработчика исключений.

Группа элементов, именуемая специальные действия, предусматривается для случаев моделирования различного поведения. К указанной группе относятся элементы создания и разрушения объектов и связей, чтения и записи переменных, передачи сигнала, приема события.

В диаграммах деятельности сервис-ориентированных средств описанные компоненты модели будут представляться в виде определенных типов графических элементов языка UML 2.0.

Вводимые расширения основы моделирования интеграции сервис-ориентированных средств касаются отображения статистических свойств действий в деятельности и потока управления, координирующего их последовательность.

В дальнейшем статистические свойства действий сервис-ориентированных средств рассматриваются во временном контексте, поскольку фактор времени является одним из основных при определении качества профессиональной деятельности в среде сервис-ориентированной архитектуры.

При отображении статистических свойств учитывается дискретность времени, являющаяся следствием цифрового характера технологий сервис-ориентированной архитектуры.

В соответствии с вышеизложенным каждое действие любого из сервис-ориентированных средств описывается плотностью вероятности $u_i(k_i), k_i = 1,2,...,K_i$

$$\sum_{k_i}^{K_i} u_i(k_i) = 1, i = 1,2,...,I\ , \qquad (1)$$

где k_i – дискретное время выполнения *i*-го действия, K_i – верхняя граница дискретного времени выполнения *i*-го действия, i – номер некоторого действия; I – общее число действий.

Плотность вероятности указывается внутри графического элемента узла действия.

Вводимые расширения относятся и к потоку управления. На дугах, исходящих из каждого узла решения, указываются $p_{j,l}, j = 1,2,...,J; l = 1,2,...,L_j$ вероятности выбора альтернативных вариантов поведения в ходе деятельности, которые удовлетворяют условию полной группы несовместных событий:

$$\sum_{l=1}^{L_j} p_{j,l} = 1,\ j = 1,2,...,J\ , \qquad (2)$$

где j – номер узла решения; L_j – число альтернативных вариантов поведения после решения j, J – число узлов решения.

Формирование модели интеграции сервис-ориентированных средств осуществляется в результате выполнения следующих шагов:

1. Позиционирование вида деятельности в среде сервис-ориентированной архитектуры.
2. Выделение множества действий D ($|D|=I$) в деятельности сервис-ориентированных средств.
3. Описание каждого действия $d_i, i = 0,1,2,...,I$ плотностью вероятности $u_i(k_i), k_i = 1,2,...,K_i$, удовлетворяющей условию (1). Плотность вероятности $u_i(k_i), k_i = 1,2,...,K_i$ может определяться посредством импорта результатов работы системы мониторинга в инфраструктуре гетерогенной сети либо путем анализа конечной цепи Маркова, марковского или полумарковского процесса, основные этапы которого раскрываются в [7].
4. Формирование потока управления из узлов координации действий сервис-ориентированных средств.
5. Описание каждого альтернативного варианта всех узлов решения соответствующей вероятностью $p_{j,l}, j = 1,2,...,J; l = 1,2,...,L_j$ с обязательным выполнением условия (2). Указанные вероятности могут определяться с помощью считывания соответствующих результатов работы системы мониторинга или оценки вероятности событий, происходящих при выполнении действий в деятельности сервис-ориентированных средств. Если любой из указанных путей определения признаётся не приемлемым, то вероятности альтернативного выбора считаются варьируемыми параметрами, принимающими случайные значения на интервале [0,1] при выполнении условия (2).
6. Формирование матрицы инциденций для узлов разъединения и узлов соединения **A** размера ($n \times n$), где n – общее число узлов разъединения и узлов соединения; $a_{i,j} = 0$, если узлы не связаны через узлы действий; $a_{i,j} = 1$, если j-ому узлу предшествуют узлы действий, следующие в последовательности узлов после i-ого узла; $a_{i,j} = -1$, если узлы действий, предшествующие i-ому узлу, следуют после j-ого узла.

7. Описание спецификаций всех узлов соединений, характеризующих взаимодействие сервис-ориентированных средств.

5.4.1. Показатели качества совместной работы служб и методы их оценки

В соответствии с концепцией сервис-ориентированной архитектуры принимается гипотеза о взаимной независимости времен выполнения действий в деятельности.

Стохастический профиль интеграции сервис-ориентированных средств в полной мере характеризуется плотностью вероятности времени выполнения деятельности, вследствие чего она выбирается базовым показателем качества совместной работы служб. После нахождения базового показателя качества без особых вычислительных затруднений определяются показатели, представляющие собой числовые характеристики времени выполнения деятельности в среде сервис-ориентированной архитектуры. При выдвижении временного ограничения, обусловленного спецификой профессиональной деятельности, наибольшее внимание уделяется риску срыва временного регламента. Считается, что чем ниже риск, тем выше качество совместной работы служб. По этой причине риск срыва временного регламента деятельности также включается в систему показателей качества.

Для определения выбранных показателей качества совместной работы служб предлагается метод, содержащий следующие этапы:

1. Выделение в модели интеграции сервис-ориентированных средств последовательностей узлов действий, замена каждой последовательности новым узлом более сложного действия с определением эквивалентной характеристики в виде плотности вероятности времени его выполнения по следующей формуле:

$$u(k_{0,1,\ldots,m}) = \sum_{\min k_{0,1,\ldots,(m-1)}}^{\max k_{0,1,\ldots,(m-1)}} u(k_{0,1,\ldots,(m-1)}) u_m(k_{0,1,\ldots,m} - k_{0,1,\ldots,(m-1)}), \quad (3)$$

$$k_{0,1,\ldots,m} = \min(k_0 + k_1 + \ldots + k_m), \ldots, \max(k_0 + k_1 + \ldots + k_m), m = 0,1,\ldots,M_j,$$

$$u(k_0) = u_0(k_0),$$

где $k_{0,1,\ldots,m}$ – дискретное время выполнения последовательности (m+1) действий; $u(k_{0,1,\ldots,m})$ – плотность вероятности времени выполнения последовательности (m+1) действий.

2. Нахождение в модели интеграции сервис-ориентированных средств группы узлов альтернативных действий, замена каждой найденной группы новым узлом более сложного действия с определением эквивалентной характеристики в виде плотности вероятности времени его выполнения согласно соотношению (4):

$$u(k_{1,2,...,l,...,L_j}) = \sum_{l=1}^{L_j} p_{j,l} u_l(k_l), \quad (4)$$

$$k_{1,2,...,l,...,L_j} = \min_l k_l, ..., \max_l k_l;\ l = 1,2,...,L_j;$$

где $u(k_{1,2,...,l,...,L_j})$ – плотность вероятности $k_{1,2,...,l,...,L_j}$ времени выполнения L_j альтернативных действий.

3. Выделение последовательностей узлов новых более сложных действий, замена каждой выделенной последовательности новым узлом укрупненного действия с определением по формуле (3) эквивалентной характеристики в виде плотности вероятности его выполнения.
4. Представление спецификаций узлов соединений модели интеграции сервис-ориентированных средств в базисе функций $\wedge(N)$, $\vee(N)$, "M из N", где N – степень параллельности, M – число выполненных действий, по окончании которых завершается соединение параллельных действий.
5. Выделение в модели групп узлов параллельных действий, замена каждой группы новым узлом укрупненного действия с определением эквивалентной характеристики в виде плотности вероятности времени его выполнения по формуле (5), (6), если соединение осуществляется согласно булевой функции $\wedge(N)$, или по формуле (7), (8), если узел соединения описывается булевой функцией $\vee(N)$, или по формулам (9), (10), (11), (12), если соединение проводится в соответствии с функцией "M из N".

$$u_{\wedge}(k_{1,2,...,n,...,N}) = \prod_{n=1}^{N} \left(\sum_{k_n=1}^{k_{1,2,...,n,...,N}} u_n(k_n) \right) - \prod_{n=1}^{N} \left(\sum_{k_n=1}^{k_{1,2,...,n,...,N}-1} u_n(k_n) \right), \quad (5)$$

$$k_{1,2,...,n,...,N} = \max_n (\min k_1, \min k_2, ..., \min k_n, ..., \min k_N), ...,$$
$$..., \max_n (\max k_1, \max k_2, ..., \max k_n, ..., \max k_N), \quad (6)$$

$$u_{\vee}(k_{1,2,...,n,...,N}) = \prod_{n=1}^{N} \left(1 - \sum_{k_n=1}^{k_{1,2,...,n,...,N}-1} u_n(k_n)\right) - \prod_{n=1}^{N} \left(1 - \sum_{k_n=1}^{k_{1,2,...,n,...,N}} u_n(k_n)\right), \quad (7)$$

$$k_{1,2,\ldots,n,\ldots,N} = \min_{n}(\min k_1, \min k_2, \ldots, \min k_n, \ldots, \min k_N), \ldots, \\ \ldots, \min_{n}(\max k_1, \max k_2, \ldots, \max k_n, \ldots, \max k_N) \quad , (8)$$

$$u_{M,N}(k_{1,2,\ldots,n,\ldots,N}) = U_{M,N}(k_{1,2,\ldots,n,\ldots,N}) - U_{M,N}(k_{1,2,\ldots,n,\ldots,N} - 1), \quad (9)$$

$$U_{M,N}(k_{1,2,\ldots,n,\ldots,N}) = \sum_{r=1}^{k_{1,2,\ldots,n,\ldots,N}} u_{\wedge}(r) \; при \; M = N, \quad (10)$$

$$U_{M,N}(k_{1,2,\ldots,n,\ldots,N}) = \sum_{r=1}^{k_{1,2,\ldots,n,\ldots,N}} u_{\vee}(r) \; при \; M = 1, \quad (11)$$

$$U_{M,N}(k_{1,2,\ldots,n,\ldots,N}) = G(N, M, N, k_{1,2,\ldots,n,\ldots,N}) \; при \; 1 < \mathrm{M} < \mathrm{N}, \quad (12)$$

где

$$G(N, M, IND, k_{1,2,\ldots,n,\ldots,N}) = \begin{cases} 0, \; если \; M > N; \\ \sum_{r=1}^{k_{1,2,\ldots,n,\ldots,N}} u_{\vee}(r), \; если \; M = 1; \\ \sum_{r=1}^{k_{1,2,\ldots,n,\ldots,N}} u_{\wedge}(r), \; если \; M = N; \\ U_{IND}(k_{1,2,\ldots,n,\ldots,N}) G(N-1, M-1, IND-1, k_{1,2,\ldots,n,\ldots,N}) + \\ +(1 - U_{IND}(k_{1,2,\ldots,n,\ldots,N})) \times \\ \times G(N-1, M, IND-1, k_{1,2,\ldots,n,\ldots,N}), \; если \; M < N; \end{cases}$$

$$U_{IND}(k_{1,2,\ldots,n,\ldots,N}) = \sum_{r=1}^{k_{1,2,\ldots,n,\ldots,N}} u_{IND}(r),$$

$$IND = 1, 2, \ldots, N;$$

$k_{1,2,\ldots,n,\ldots,N}$ – дискретное время выполнения параллельных действий;

$u_{\wedge}(k_{1,2,\ldots,n,\ldots,N})$ – плотность вероятности дискретного времени выполнения параллельных действий при соединении согласно булевой функции $\wedge(N)$;

$u_{\vee}(k_{1,2,\ldots,n,\ldots,N})$ – плотность вероятности дискретного времени выполнения параллельных действий при соединении согласно булевой функции $\vee(N)$;

$u_{M,N}(k_{1,2,\ldots,n,\ldots,N})$ – плотность вероятности дискретного времени выполнения параллельных действий при соединении согласно функции "M из N";

$U_{M,N}(k_{1,2,\ldots,n,\ldots,N})$ – функция распределения дискретного времени выполнения параллельных действий при соединении согласно функции "*M* из *N*".

6. Формирование последовательности узлов укрупненных действий и определение $u(k_{0,1,\ldots,i,\ldots,I})$ плотности вероятности времени выполнения деятельности согласно соотношению (3).
7. Определение показателей качества совместной работы служб

$$E[k_{0,1,\ldots,i,\ldots,I}] = \sum_{\min k_{0,1,\ldots,i,\ldots,I}}^{\max k_{0,1,\ldots,i,\ldots,I}} k_{0,1,\ldots,i,\ldots,I} u(k_{0,1,\ldots,i,\ldots,I}), \quad (13)$$

$$D[k_{0,1,\ldots,i,\ldots,I}] = \sum_{\min k_{0,1,\ldots,i,\ldots,I}}^{\max k_{0,1,\ldots,i,\ldots,I}} (k_{0,1,\ldots,i,\ldots,I} - E[k_{0,1,\ldots,i,\ldots,I}])^2 u(k_{0,1,\ldots,i,\ldots,I}), \quad (14)$$

$$R(k_{0,1,\ldots,i,\ldots,I} > C) = 1 - \sum_{\min k_{0,1,\ldots,i,\ldots,I}}^{C} u(k_{0,1,\ldots,i,\ldots,I}), \quad (15)$$

где C – верхняя граница допустимого времени выполнения деятельности; $E[k_{0,1,\ldots,i,\ldots,I}]$, $D[k_{0,1,\ldots,i,\ldots,I}]$ – соответственно математическое ожидание и дисперсия времени выполнения деятельности в среде сервис-ориентированной архитектуры; $R(k_{0,1,\ldots,i,\ldots,I} > C)$ – риск срыва временного регламента.

Для подтверждения корректности определения показателей качества совместной работы служб целесообразно воспользоваться модифицированным методом свертки, раскрытым в [7].

Исходным материалом для применения модифицированного метода свертки будет являться логическая модель деятельности, которая формируется посредством преобразования диаграммы деятельности. Преобразование осуществляется путем замены узлов действий и узлов решений неузловыми вершинами ориентированного графа, а узлов разъединения и узлов соединения узловыми вершинами. При использовании модифицированного метода вырожденный граф строится по матрице инциденций **A**, сформированной при разработке диаграммы деятельности.

Прикладной аспект разработанных формализаций распространяется, прежде всего, на типовые профили интеграции сервис-ориентированных средств, широко востребованных в профессиональной деятельности. При подобной направленности применения формализаций обеспечивается формирование рациональных профилей услуг в зависимости от условий выполнения профессиональной деятельности.

Далее ключевые особенности определения показателей качества совместной работы служб раскрываются при анализе типовых профилей интеграции сервис-ориентированных средств.

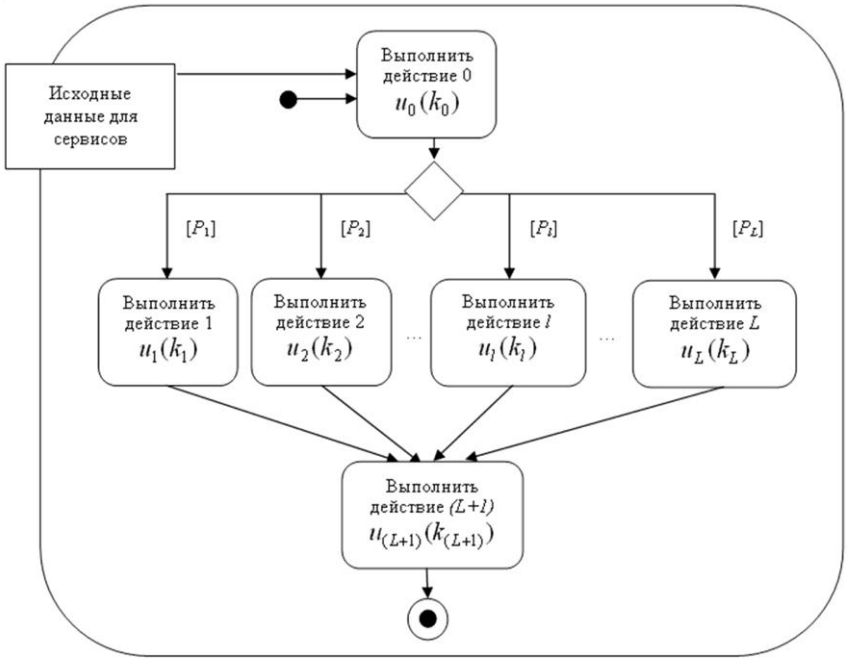


Рис. 47. Типовая модель альтернативных действий

На рис. 47 приводится модель сервисной деятельности в случае, когда запрос клиента может выполняться одним из L сервис-ориентированных средств, развернутых на серверах с различными характеристиками производительности и сконфигурированных в среде исполнения крупно-гранулярных процессов. В рассматриваемой модели действие 0 ассоциируется с инициализацией сервисной деятельности, последующие L действий – с альтернативным выполнением сервисов, а завершающее (L+1) действие – с обработкой данных и формированием ответа клиенту. Клиентской стороной ставится цель выполнения сервисной деятельности и получения информации относительно ее качества, а администратором гетерогенной сети – балансировка нагрузки сервисного обслуживания. Альтернативные действия в рассматриваемой

схеме технически представляют собой сетевые адреса точек доступа к веб-сервису, развернутому на L серверах.

В соответствии с предложенным методом математическое ожидание, дисперсия времени реализации сервисной деятельности и риск срыва временного регламента определяются с помощью соотношений (16) – (21):

$$u(k_{1,2,...,l,...,L}) = \sum_{l=1}^{L} p_l u_l(k_l), \quad (16)$$

$$k_{1,2,...,l,...,L} = \min_l k_l, ..., \max_l k_l; \ l = 1,2,...,L_j;$$

$$u(k_{0,1,2,...,l,...,L}) = \sum_{k_0}^{K_0} u_0(k_0) u(k_{0,1,2,...,l,...,L} - k_0), \quad (17)$$

$$k_{0,1,...,l} = \min(k_0 + k_{1,2,...,l}), ..., \max(k_0 + k_{1,2,...,l}), l = 1,..., L;$$

$$u(k_{0,1,...,l,...,L,(L+1)}) = \sum_{\min k_{0,1,...,l,...,L}}^{\max k_{0,1,...,l,...,L}} u(k_{0,1,...,l,...,L}) u_{(L+1)}(k_{0,1,...,l,...,L,(L+1)} - k_{0,1,...,l,...,L}), \quad (18)$$

$$k_{0,1,...,l,...,L,(L+1)} = \min(k_{0,1,2,...,l,...,L} + k_{(L+1)}), ..., \max(k_{0,1,2,...,l,...,L} + k_{(L+1)});$$

$$E[k_{0,1,...,l,...,L,(L+1)}] = \sum_{\min k_{0,1,...,l,...,L,(L+1)}}^{\max k_{0,1,...,l,...,L,(L+1)}} k_{0,1,...,l,...,L,(L+1)} u(k_{0,1,...,l,...,L,(L+1)}); \quad (19)$$

$$D[k_{0,1,...,l,...,L,(L+1)}] = \\ = \sum_{\min k_{0,1,...,l,...,L,(L+1)}}^{\max k_{0,1,...,l,...,L,(L+1)}} (k_{0,1,...,l,...,L,(L+1)} - E[k_{0,1,...,l,...,L,(L+1)}])^2 \times u(k_{0,1,...,l,...,L,(L+1)}); \quad (20)$$

$$R(k_{0,1,...,l,...,L,(L+1)} > C) = 1 - \sum_{\min k_{0,1,...,l,...,L,(L+1)}}^{C} u(k_{0,1,...,l,...,L,(L+1)}). \quad (21)$$

При реализации работы служб на языке BPEL в среде NetBeans 6.5 в соответствии с типовой моделью альтернативных действий представление крупно-гранулярных процессов формируются согласно схеме с рис. 48.

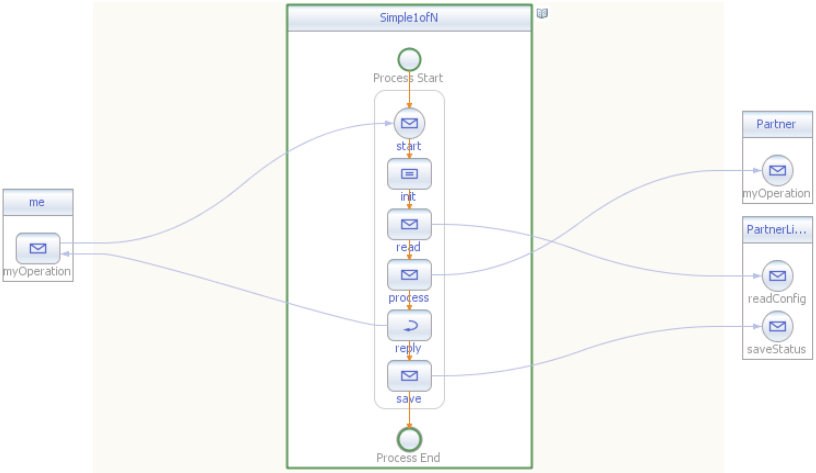


Рис. 48. Типовая модель альтернативных действий на языке BPEL

На практике параллельная работа служб именуется реализацией партнерских сервисов. Она представлена на диаграмме деятельности на рис. 49.

При вариации функциональной спецификации узла соединения образуется ряд типовых профилей интеграции сервис-ориентированных средств, базовыми среди которых являются варианты, описываемые функциями $\wedge(N)$, $\vee(N)$, "M из N".

Первые два из указанных профилей являются частными случаями третьего варианта: функция $\wedge(N)$ соответствует функции "M из N" при M=N, а функции $\vee(N)$ – "M из N" при M=1.

В том случае, когда для успешного выполнения исследуемой деятельности требуется информация от всех параллельно реализуемых партнерских сервисов, то спецификация соединения описывается булевой функцией $\wedge(N)$. В подобной ситуации партнерские сервисы сопровождаются отличающимися интерфейсами и предназначаются для выполнения различных функций, подчиненных достижению единой цели. Примером такого рода может являться широкий спектр крупно-гранулярных процессов (бизнес-процессов), в которых обработка отдельных частей входной информации проводится несколькими партнерами. Например, при оказании услуги оплаты определенных видов Интернет-контента со счёта мобильного оператора проводится несколько

действий, среди которых авторизация абонента, получение информации о предоставляемом Интернет-контенте (профиль услуги).

Если для успешного выполнения исследуемой деятельности требуется информация от любого первого завершенного партнерского сервиса из N реализуемых, то спецификация соединения представляется булевой функцией $\vee(N)$. Подобного рода ситуация проявляется тогда, когда среди N предусматриваемых авторизаций неудачно завершается хотя бы любая одна.

В случае, когда у партнерских сервисов одинаковые интерфейсы и для успешного выполнения исследуемой деятельности оказывается достаточным завершения M опрашиваемых средств из N параллельно реализуемых, узел соединения описывается функцией "M из N". Подобная ситуация наблюдается при поиске контента на множестве серверов. Например, в крупно-гранулярном процессе, осуществляющем поиск ссылок на видеоконтент на нескольких серверах контент-провайдеров, в параллельных потоках вызывается N партнерских сервисов поиска.

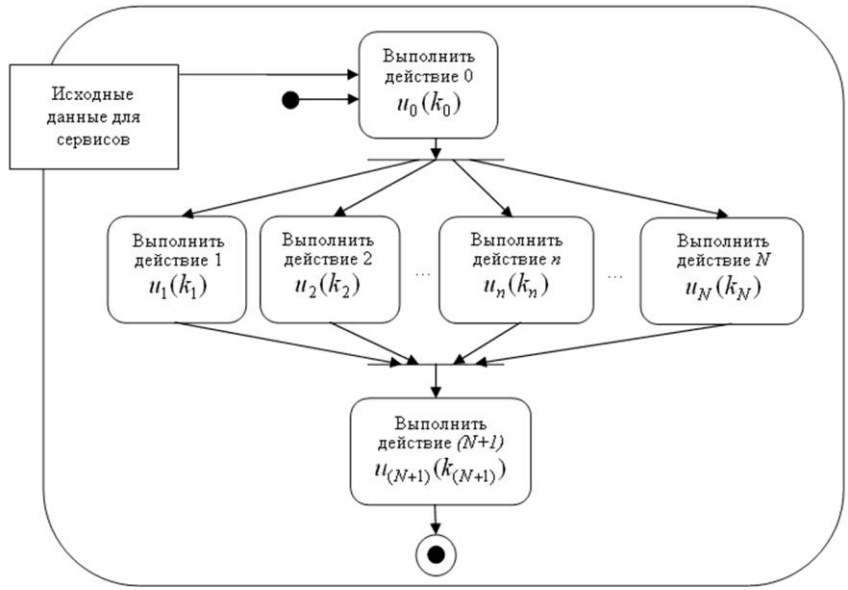


Рис. 49. Типовая модель параллельных действий

При описании узла соединения спецификацией $\wedge(N)$ математическое ожидание, дисперсия времени совместной работы служб и риск срыва регламента определяются соответственно соотношениями (5), (6), (22) – (26),

$$u(k_{0,1,2,\ldots,n,\ldots,N}) = \sum_{k_0}^{K_0} u_0(k_0) u_{\wedge}(k_{0,1,2,\ldots,n,\ldots,N} - k_0), \quad (22)$$

$$k_{0,1,\ldots,n} = \min(k_0 + k_{1,2,\ldots,n}), \ldots, \max(k_0 + k_{1,2,\ldots,n}), n = 1,2,\ldots,N;$$

$$u(k_{0,1,\ldots,n,\ldots,N,(N+1)}) = \\ = \sum_{\min k_{0,1,\ldots,n,\ldots,N}}^{\max k_{0,1,\ldots,n,\ldots,N}} u(k_{0,1,\ldots,n,\ldots,N)}) u_{(N+1)}(k_{0,1,\ldots,n,\ldots,N,(N+1)} - k_{0,1,\ldots,n,\ldots,N}), \quad (23)$$

$$k_{0,1,\ldots,n,\ldots,N,(N+1)} = \min(k_{0,1,\ldots,n,\ldots,N} + k_{(N+1)}), \ldots, \max(k_{0,1,\ldots,n,\ldots,N} + k_{(N+1)});$$

$$E[k_{0,1,\ldots,n,\ldots,N,(N+1)}] = \sum_{\min k_{0,1,\ldots,n,\ldots,N,(N+1)}}^{\max k_{0,1,\ldots,n,\ldots,N,(N+1)}} k_{0,1,\ldots,n,\ldots,N,(N+1)} u(k_{0,1,\ldots,n,\ldots,N,(N+1)}); \quad (24)$$

$$D[k_{0,1,\ldots,n,\ldots,N,(N+1)}] = \\ = \sum_{\min k_{0,1,\ldots,n,\ldots,N,(N+1)}}^{\max k_{0,1,\ldots,n,\ldots,N,(N+1)}} (k_{0,1,\ldots,n,\ldots,N,(N+1)} - E[k_{0,1,\ldots,n,\ldots,N,(N+1)}])^2 \times u(k_{0,1,\ldots,n,\ldots,N,(N+1)}); \quad (25)$$

$$R(k_{0,1,\ldots,n,\ldots,N,(N+1)} > C) = 1 - \sum_{\min k_{0,1,\ldots,n,\ldots,N,(N+1)}}^{C} u(k_{0,1,\ldots,n,\ldots,N,(N+1)}). \quad (26)$$

Если узел соединения характеризуется функцией $\vee(N)$, то числовые характеристики времени совместной работы служб и риск срыва регламента находятся с помощью выражений (7), (8), (27), (23) – (26):

$$u(k_{0,1,2,\ldots,n,\ldots,N}) = \sum_{k_0}^{K_0} u_0(k_0) u_{\vee}(k_{0,1,2,\ldots,n,\ldots,N} - k_0), \quad (27)$$

$$k_{0,1,\ldots,n} = \min(k_0 + k_{1,2,\ldots,n}), \ldots, \max(k_0 + k_{1,2,\ldots,n}), n = 1,2,\ldots,N \cdot$$

Когда спецификация узла соединения представляется функцией "M из N", тогда статистические показатели времени совместной работы служб и риск срыва регламента вычисляются по формулам (9) – (12), (28), (23) – (26):

$$u(k_{0,1,2,\ldots,n,\ldots,N}) = \sum_{k_0}^{K_0} u_0(k_0) u_{M,N}(k_{0,1,2,\ldots,n,\ldots,N} - k_0), \quad (28)$$

$$k_{0,1,\ldots,n} = \min(k_0 + k_{1,2,\ldots,n}), \ldots, \max(k_0 + k_{1,2,\ldots,n}), n = 1,\ldots,N \cdot$$

При реализации совместной работы служб на языке BPEL в среде NetBeans 6.5 последние три типовых варианта крупно-гранулярных процессов формируются согласно схеме с рис. 50.

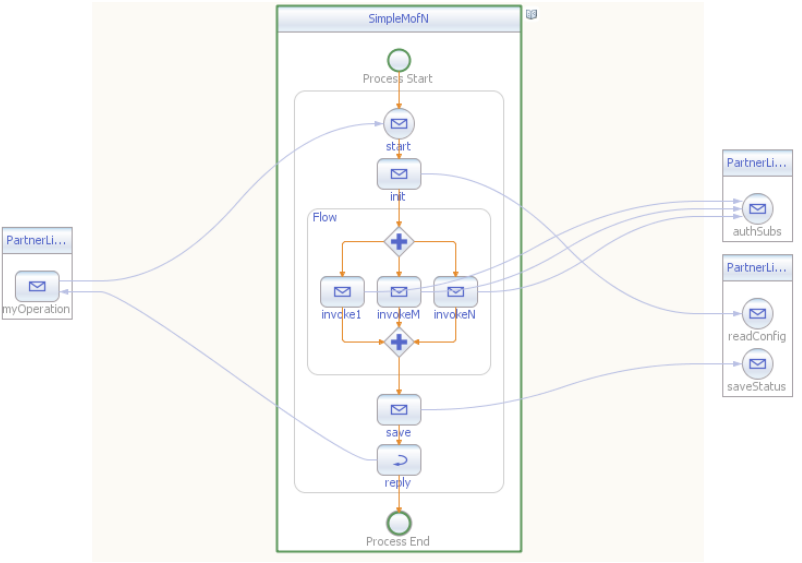


Рис. 50. Типовая модель параллельных действий на языке BPEL

5.5. Кодирование веб-сервиса

При создании веб-сервиса разработчику, как правило, не нужно манипулировать непосредственно SOAP-сообщениями: существует множество фреймворков, позволяющих скрыть обработку XML-данных веб-сервисов с помощью абстракций, характерных для языка программирования, на котором ведется разработка. Подобные фреймворки значительно упрощают процесс разработки, скрывая за собой целый стек протоколов, имплементирующих стандарты в области построения и обмена сообщениями веб-сервисами, описанные в [8]. Согласно [9, 10] одной из передовых программных систем подобного рода для языка Java является стек протоколов JAX-WS корпорации Sun, который будет использован в описании технологического процесса.

Краткий обзор новых технических возможностей версии 2 данной системы приведен в [11].

Различают несколько подходов к разработке веб-сервисов и использованию фреймворка на основе стека JAX-WS:

- разработка от WSDL (сначала создается WSDL-описание, по которому генерируется java-код имплементации);
- разработка от java-кода (создается Java-класс с соответствующими аннотациями, определяющими его как имплементацию веб-сервиса; WSDL-описание генерируется автоматически в момент развертывания приложения).

Рассмотрим один из наиболее быстрых и эффективных способов разработки веб-сервиса в среде NetBeans – от java-кода. Более подробный материал обо всех подходах изложен в [12].

5.5.1. Создание веб-приложения

Через меню "File>New Project…" необходимо создать проект веб-приложения с именем "AuthorizeWebServiceApp". В мастере создания нового проекта должен быть выбран установленный сервер приложений Glassfish и домен "domain1" в качестве целевого сервера для развертывания приложения (см. рис. 51).

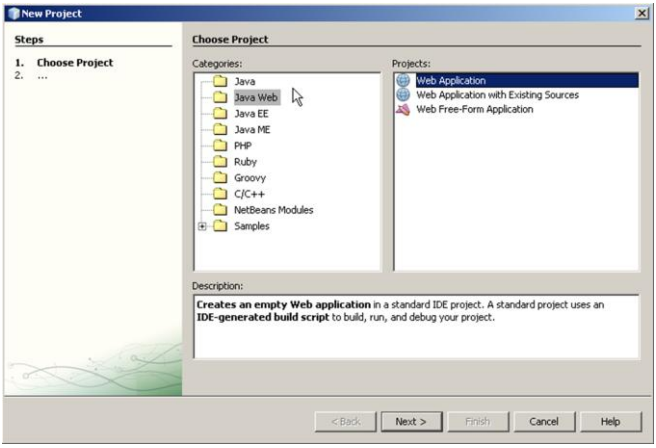


Рис. 51. Мастер создания веб-приложения

5.5.2. Имплементация сервиса

После создания проекта веб-приложения необходимо добавить имплементацию веб-сервиса следующими действиями:

1. Добавить новый java-класс с именем AuthWS в пакет com.my.ws.
2. Добавить аннотацию @WebService перед объявлением класса.
3. Создать функцию authorize() с выходным значением типа boolean и с входными параметрами:
    a. name типа String,
    b. password типа String.
4. Имплементировать бизнес-логику проверки имени авторизируемого пользователя. В рассматриваемом примере достаточным условием успешной авторизации является совпадение строк имени пользователя и его пароля с заранее заданными значениями "Alice" и "inwonderland", как это представлено на рис. 52.
5. Добавить аннотацию @WebMethod перед объявлением функции authorize().
6. Добавить аннотации @WebParam(name="name"), @WebParam(name="password") перед параметрами функции.

```
1   package com.my.ws;
2
3   @javax.jws.WebService
4   public class AuthWS {
5       @javax.jws.WebMethod
6       public boolean authorize(
7               @javax.jws.WebParam(name="name")
8               String name,
9               @javax.jws.WebParam(name="password")
10              String password) {
11          return "Alice".equals(name) &&
12                  "inwonderland".equals(password);
13      }
14  }
```

Рис. 52. Пример имплементации веб-сервиса авторизации

5.5.3. Сборка и развертывание веб-сервиса

Сборка приложения происходит автоматически в момент развертывания веб-приложения на сервере, указанном в качестве целевого для данного проекта. Предварительно необходимо запустить домен, где будет произведено развертывание командой утилиты управления, входящей в состав Glassfish:

```
> asadmin start-domain domain1
```

Развертывание приложения из среды разработки NetBeans производится выполнением команды Deploy из контекстного меню проекта веб-приложения, при этом URL-адрес точки доступа к сервису выводится в окно трассировки сообщений сервера приложений (пункт меню Window>Output>Output). Если настройки проекта не были изменены, то рассматриваемый пример сервиса развертывается по адресу:

http://localhost:8080/AuthorizeWebServiceApp/AuthWSService

WSDL-интерфейс веб-сервиса генерируется в момент развертывания автоматически и становится доступен по адресу:

http://localhost:8080/AuthorizeWebServiceApp/AuthWSService?wsdl

Для развертывания приложения на нескольких серверах в сети, например, в целях нагрузочного тестирования, эффективнее использовать возможность среды NetBeans собирать проект отдельным WAR-файлом, который впоследствии копируется из каталога dist, создаваемого во время сборки.

5.5.4. Тестирование сервиса

Наиболее продвинутым инструментом тестирования сервис-ориентированных средств среди бесплатно распространяемых продуктов с открытым исходным кодом является программный комплекс SoapUI (soapui.org) от компании Eviware [13]. Он доступен для копирования по адресу:

http://sourceforge.net/projects/soapui/files/soapui/

Для разработки функциональных тестов сервиса необходимо создать проект через пункт меню File>New SoapUI Project, указав при этом имя проекта и путь или URL-адрес WSDL-описания сервиса. Затем производятся следующие действия:

- создается тестовый набор (контекстное меню проекта New TestSuite);
- создается тестовый сценарий (контекстное меню тестового набора New TestCase);

- в тестовый сценарий добавляется шаг теста, вызывающий веб-сервис (Add Step>SOAP Test Request):
    - при создании данного шага целесообразно указать в мастере необходимость валидации ответного сообщения на соответствие XSD-схеме (Add Schema Assertion) и на отсутствие сообщения об ошибке (Add Not SOAP Fault Assertion);
    - необходимо выбрать пункт "Create optional elements", чтобы ускорить редактирование SOAP-запроса;
    - в открывшемся редакторе SOAP-запроса (см. рис. 53) необходимо ввести тестовые данные: имя и пароль пользователя (в качестве примера было создано 2 шага сценария с различными тестовыми данными:
        - Alice/inwonderland (авторизация успешна),
        - User/unknown (авторизация неуспешна));
    - проверку выходного значения для каждого шага теста возможно выполнять автоматически, если в разделе Assertions формы редактирования запроса добавить проверку Add Assertion>Contains, где необходимо ввести текст, наличие которого будет проверено в SOAP-ответе (в примере происходит проверка на вхождение строки true или false в зависимости от шага теста).

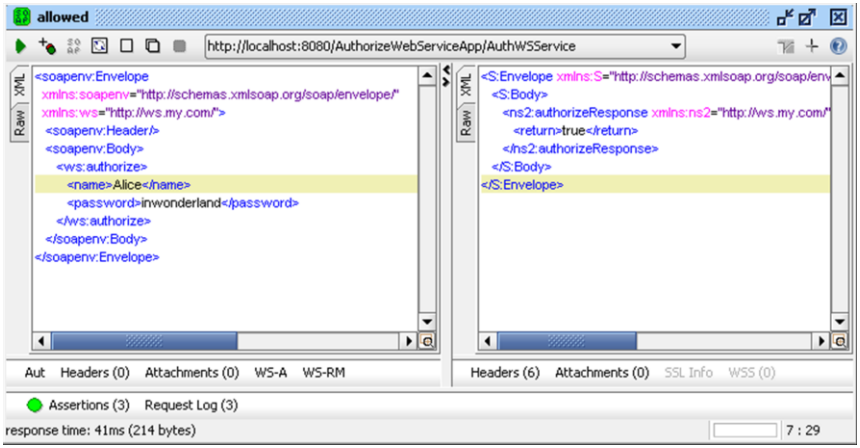


Рис. 53. Форма редактирования SOAP-запроса и просмотра SOAP-ответа

- для выполнения произвольного количества созданных шагов теста можно воспользоваться формой запуска сценария, как это проиллюстрировано на рис. 54.

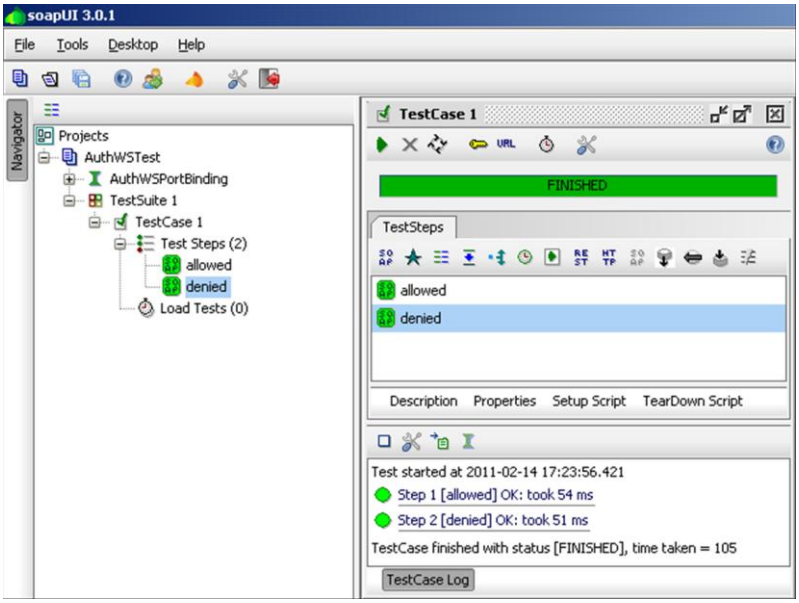


Рис. 54. Форма запуска тестового сценария

5.6. Разработка бизнес-процесса

5.6.1. Создание BPEL-проекта и импорт окружения

Разработка описанного бизнес-процесса начинается с создания BPEL проекта с именем "HelloBpel" в среде разработки NetBeans (меню File>New Project…) как это изображено на рис. 55.

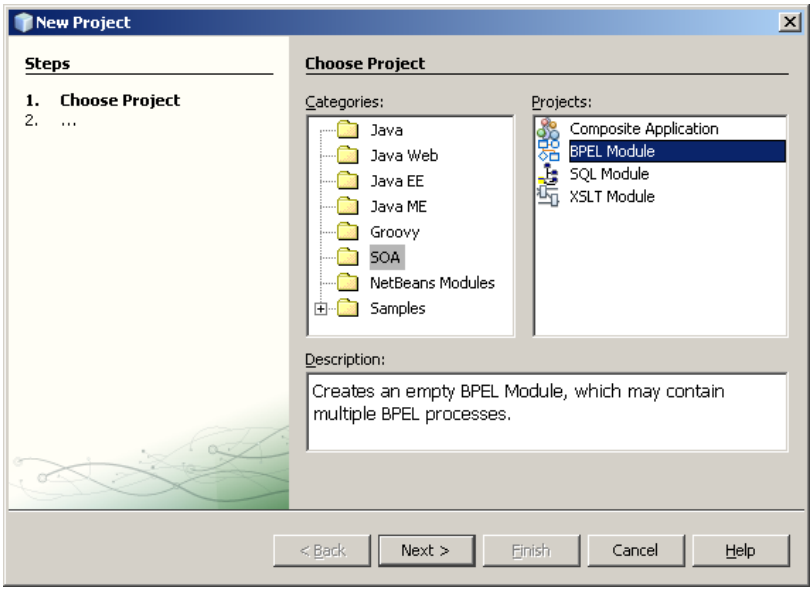


Рис. 55. Мастер создания BPEL проекта в среде NetBeans

Далее необходимо экспортировать в проект WSDL-документ созданного на предыдущем этапе стороннего сервиса AuthWS. Для этого следует раскрыть дерево проекта AuthorizeWebServiceApp, раскрыть узел Web Services, в контекстном меню сервиса AuthWS выбрать пункт Generate and Copy WSDL. В открывшемся окне проводника указать путь до директории HelloBpel/src. Среда разработки автоматически экспортирует необходимые файлы в проект, а именно, WSDL-описание и XSD-схему веб-сервиса. Данную операцию не обязательно производить средствами среды NetBeans, можно скопировать указанные файлы вручную либо в качестве альтернативы воспользоваться мастером экспорта WSDL-файла через меню File>New File>XML>External WSDL Document.

5.6.2. Создание интерфейса BPEL-процесса

Бизнес-процесс согласно концепции SOA является крупно-гранулярным сервисом, следовательно, он наследует все свойства сервис-ориентированного средства, включая наличие открытого интерфейса,

представленного WSDL-описанием. Следующим шагом разработки BPEL-процесса является создание описания его интерфейса. Данный шаг разработки разбивается на два действия:

- создание XSD-схемы элементов, описывающих состав входящих и исходящих SOAP-сообщений сервиса;
- создание WSDL-описания сервиса.

Следует отметить, что существует два основных типа кодирования содержимого SOAP-сообщений (всего существует 5 типов):

- RPC-ориентированный (RPC от Remote Procedure Call);
- Document-ориентированный.

Каждый из них накладывает определенные ограничения как на содержание SOAP-сообщения, так и на его описание в WSDL-документе. Ключевым отличием этих типов при создании WSDL является то, что Document-ориентированный стиль кодирования требует описания входных и выходных параметров WSDL-операции в XSD-схеме в качестве XML-элементов, RPC позволяет избежать этого.

В рассматриваемом примере используется тип кодирования Document/literal. Он разрабатывался позднее RPC-ориентированного типа, поэтому лишен некоторых важных недостатков последнего, в частности:

- существует возможность валидации SOAP-сообщения по его XSD-схеме;
- данный тип строго поддерживается стандартом, разработанным организацией WS-I (в отличие от RPC/encoded стиля кодирования);
- отсутствует избыточность SOAP-сообщения, связанная с передачей типов параметров WSDL-операции (в отличие от RPC/encoded).

Подробнее о типах кодирования и о том, какой тип предпочтительнее выбирать в тех или иных случаях, изложено в [14].

XSD-схему, используемую для описания веб-сервиса, можно создать в соответствующем разделе WSDL-описания либо отдельным документом, который необходимо впоследствии импортировать в WSDL-файле. Последний вариант предпочтительнее, так как позволяет воспользоваться визуальными средствами редактирования XML-схем, встроенными в среду разработки. Для создания отдельного файла XSD-схемы необходимо выбрать меню File>New File, где в категории XML выбрать тип XML Schema, указать имя файла схемы "helloXSD". При помощи визуального редактора в схему добавляются 2 элемента getGreeting и getGreetingResponse и наполняются соответственно входными параметрами name и password и выходным – greeting, как представлено на рис. 56 и рис. 57.

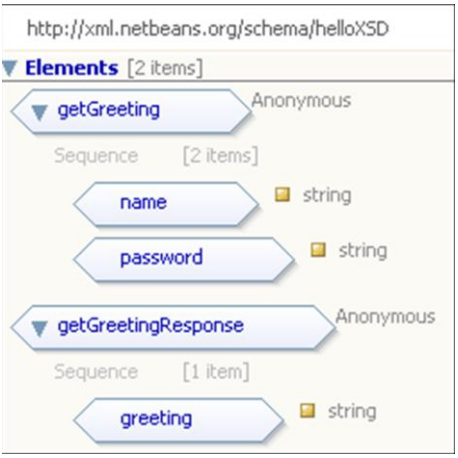


Рис. 56. Визуальный редактор XML-схемы, используемой WSDL-интерфейсом бизнес-процесса

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4      targetNamespace="http://xml.netbeans.org/schema/helloXSD"
5      xmlns:tns="http://xml.netbeans.org/schema/helloXSD"
6      elementFormDefault="qualified">
7      <xsd:element name="getGreeting">
8          <xsd:complexType>
9              <xsd:sequence>
10                 <xsd:element name="name" type="xsd:string"/>
11                 <xsd:element name="password" type="xsd:string"/>
12             </xsd:sequence>
13         </xsd:complexType>
14     </xsd:element>
15     <xsd:element name="getGreetingResponse">
16         <xsd:complexType>
17             <xsd:sequence>
18                 <xsd:element name="greeting" type="xsd:string"/>
19             </xsd:sequence>
20         </xsd:complexType>
21     </xsd:element>
22 </xsd:schema>
```

Рис. 57. Текстовое представление созданной XML-схемы бизнес-процесса

Следующим шагом разработки является создание WSDL-интерфейса BPEL-процесса. Аналогично созданию XML-схемы необходимо выбрать меню File>New File, где в категории XML отметить тип WSDL Document, а затем указать имя файла "helloWSDL". Далее необходимо импортировать файл XML-схемы, созданный на предыдущем этапе, как это проиллюстрировано на рис. 58.

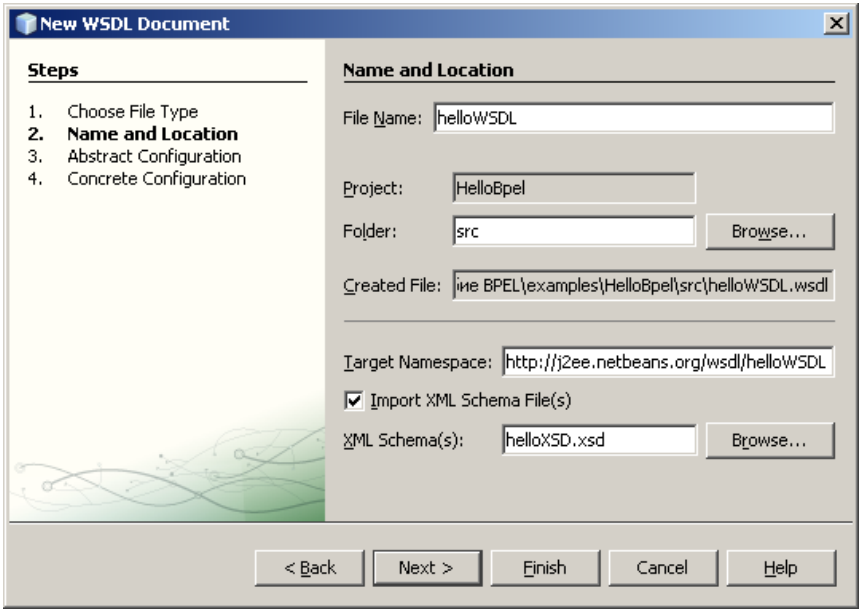


Рис. 58. Мастер создания WSDL-описания (импортирование XML-схемы)

Далее заполняется абстрактная часть конфигурации WSDL-описания, а именно, задается имя операции "getGreeting" и, согласно выбранному типу кодирования SOAP-сообщений Document\Literal, формируется содержание входящего и исходящего сообщений для данной операции.

В качестве элемента, описывающего входящее сообщение, выбирается созданный на этапе разработке XML-схемы элемент getGreeting, для исходящего сообщения – getGreetingResponse, как изображено на рис. 59.

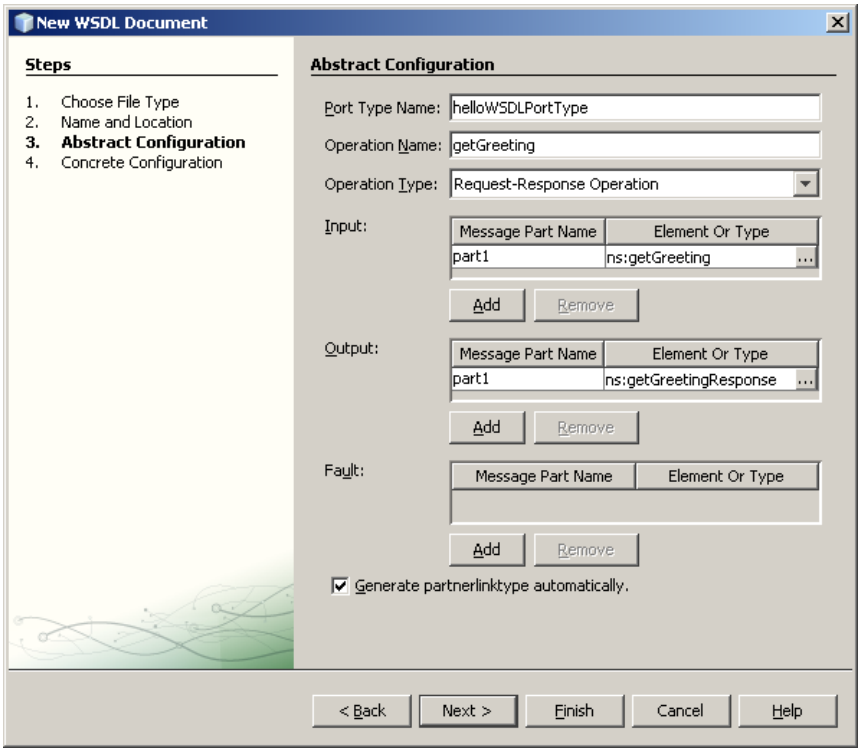


Рис. 59. Описание абстрактной части конфигурации WSDL-интерфейса

Последний шаг создания WSDL-интерфейса – заполнение конкретной части конфигурации, а именно привязка описанных сообщений к транспортному протоколу. В рассматриваемом случае привязка осуществляется к протоколу SOAP с применением Document\Literal стиля кодирования сообщения.

5.6.3. Разработка BPEL-процесса

На предыдущих этапах разработки для BPEL процесса были подготовлены его собственный WSDL-интерфейс с операцией getGreeting и импортированный интерфейс стороннего сервиса AuthWS, в контексте языка BPEL именующегося партнерским сервисом.

Непосредственное создание BPEL-процесса начинается с вызова меню File>New File, где в категории SOA необходимо выбрать тип файла BPEL Process и ввести название файла "helloBPEL".

Редактирование BPEL-процесса можно осуществлять, манипулируя исходным кодом на XML, но гораздо удобнее воспользоваться для этой цели визуальным редактором BPEL-диаграмм. Разработка в нем ведется преимущественно методом перетаскивания элементов, представляющих собой структуры языка BPEL, из палитры инструментов (запускается вызовом меню Window>Palette).

Для того, чтобы связать созданный ранее WSDL-интерфейс helloWSDL с данным BPEL-процессом, необходимо перетащить пиктограмму файла helloWSDL.wsdl из дерева проектов в поле, расположенное слева от диаграммы процесса в место, обозначенное двумя вложенными окружностями, появляющимися в момент перетаскивания. В указанном поле слева располагается произвольное количество интерфейсов, имплементируемых бизнес-процессом. В поле справа – интерфейсы партнерских сервисов, вызываемых в ходе исполнения бизнес-логики. Следующий шаг разработки, добавление партнерских связей, производится аналогично добавлению собственного интерфейса BPEL-процесса. В терминах языка BPEL любой WSDL-интерфейс моделируется партнерской связью в виде элемента PartnerLink.

Следует обратить внимание, что при добавлении партнерского WSDL в BPEL-процесс создается WSDL файл-обертка (wrapper), в котором автоматически генерируется элемент partnerLinkType с вложенным в него дочерним элементом role.

Роль необходима бизнес-процессу для того, чтобы обозначить, имплементируется ли данный интерфейс самим процессом, либо он реализован партнерским сервисом и является, по сути, сторонним интерфейсом. Подробнее об использовании ролей в языке BPEL изложено в разделе 2.2.

Событием, инициализирующим рассматриваемый бизнес-процесс, является получение входного сообщения. Для моделирования получения сообщения необходимо перетащить из палитры инструментов на диаграмму BPEL действие Receive и аналогичным образом соединить его с интерфейсом helloWSDL, потянув за пиктограмму конверта, расположенную на изображении действия Receive. Вызов команды Edit из контекстного меню данного действия открывает окно редактирования свойств, которое позволяет создать переменную BPEL-процесса GetGreetingIn, предназначенную дня помещения в неё входных параметров WSDL-операции getGreeting, как изображено на рис. 60.

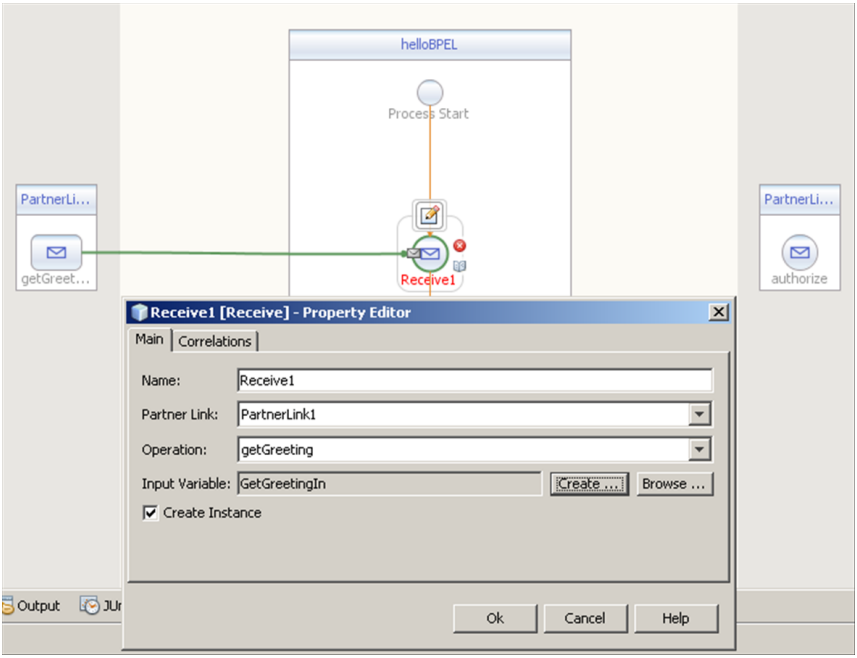


Рис. 60. Редактирование свойств действия Receive и создание переменной процесса

Каждому действию Receive двунаправленной WSDL-операции в BPEL-процессе должно соответствовать хотя бы одно действие Reply. Согласно чему, необходимо поместить на диаграмму элемент Reply и произвести аналогичные действия, выполненные при обработке входящего сообщения, иными словами, соединить блок с операцией getGreeting и создать выходную переменную.

Для того, чтобы вызвать сторонний партнерский сервис в синхронном блокирующем режиме, необходимо поместить на диаграмму действие Invoke и соединить его с партнерской связью и операцией authorize, затем так же создать входную и выходную переменные (AuthorizeIn и AuthorizeOut), содержащие параметры данной операции.

Далее необходимо проинициализировать входную переменную AuthorizeIn, для чего на BPEL-диаграмму перед действием Invoke помещается элемент Assign.

Для данного элемента доступен режим редактирования Mapper, вызываемый двойным щелчком мыши по действию Assign или нажатием

соответствующего управляющего элемента на панели инструментов BPEL-файла и предназначенный для отображения и настройки копирования одной переменной в другую, а также для проведения манипуляций с данными посредством XPath-функций. Возможности языка XPath изложены в [15].

Для копирования входных данных операции getGreeting во входные переменные операции authorize достаточно соединить соответствующие узлы структур в режиме Mapper, как изображено на рис. 61.

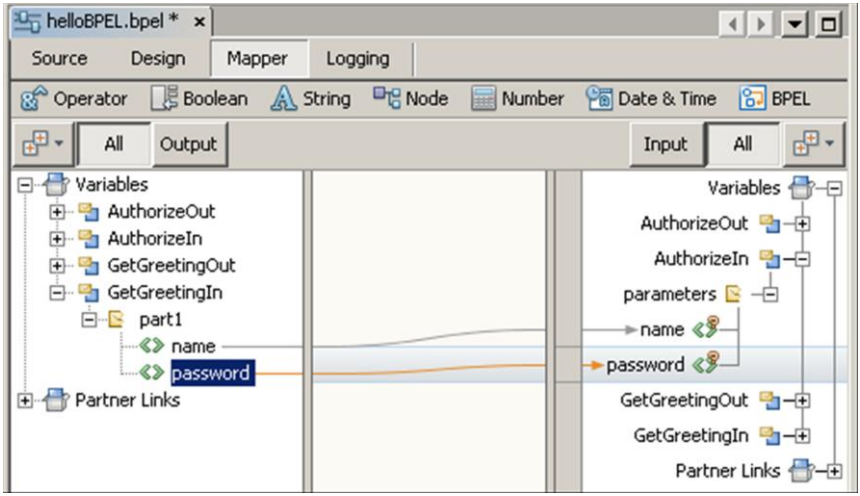


Рис. 61. Визуальный режим манипуляции дынными Mapper

Далее в режиме редактирования диаграммы BPEL (Design) следует обработать ответ операции authorize. Для этого необходимо поместить на граф бизнес-процесса структурное действие If после блока Invoke и расположить в альтернативных ветвях алгоритма действия Assign, для наглядности назвав их Allow и Deny. Для задания предиката условного ветвления следует перейти в режим Mapper и соединить узел выходного значения операции authorize с условием ветвления, как изображено на рис. 62.

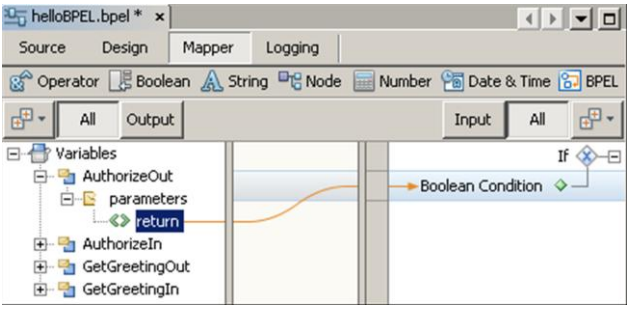


Рис. 62. Редактирование условия ветвления

Для формирования выходного сообщения приветствия в альтернативных блоках Assign с именами Allow и Deny в режиме Mapper следует присвоить переменной GetGreetingOut значение, полученное конкатенацией строк приветствия и имени пользователя. Для этого необходимо выделить курсором целевую переменную и добавить визуальное представление XPath-функции Concat, расположенное в категории String панели инструментов окна редактирования в режиме Mapper, затем соединить узлы структур данных, как проиллюстрировано на рис. 63 для действия с именем Allow.

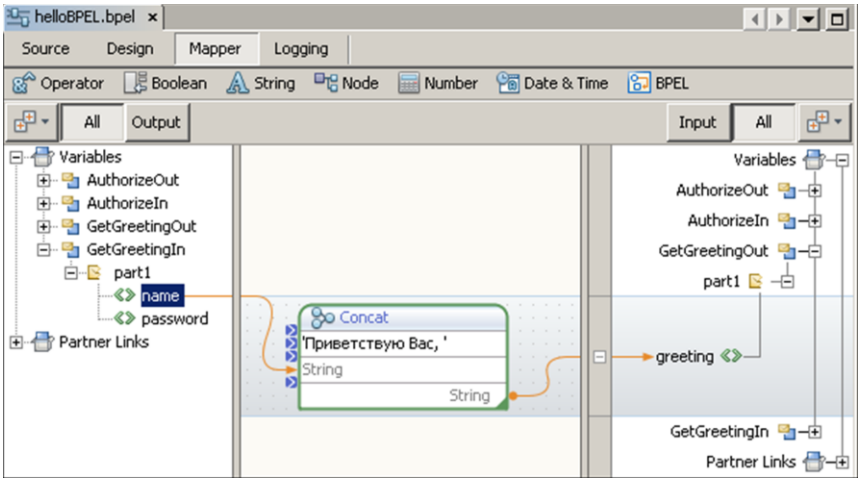


Рис. 63. Пример использования XPath-функции Concat для формирования значения переменной

Для действия Assign с именем Deny текст сообщения в функции Concat изменяется соответственно на "Доступ запрещен для ".

Разработка BPEL-процесса завершена. Результат отображен на рис. 64. Чтобы провести его валидацию и убедиться в отсутствии критичных и некритичных ошибок необходимо вызвать команду меню Run>Validate XML.

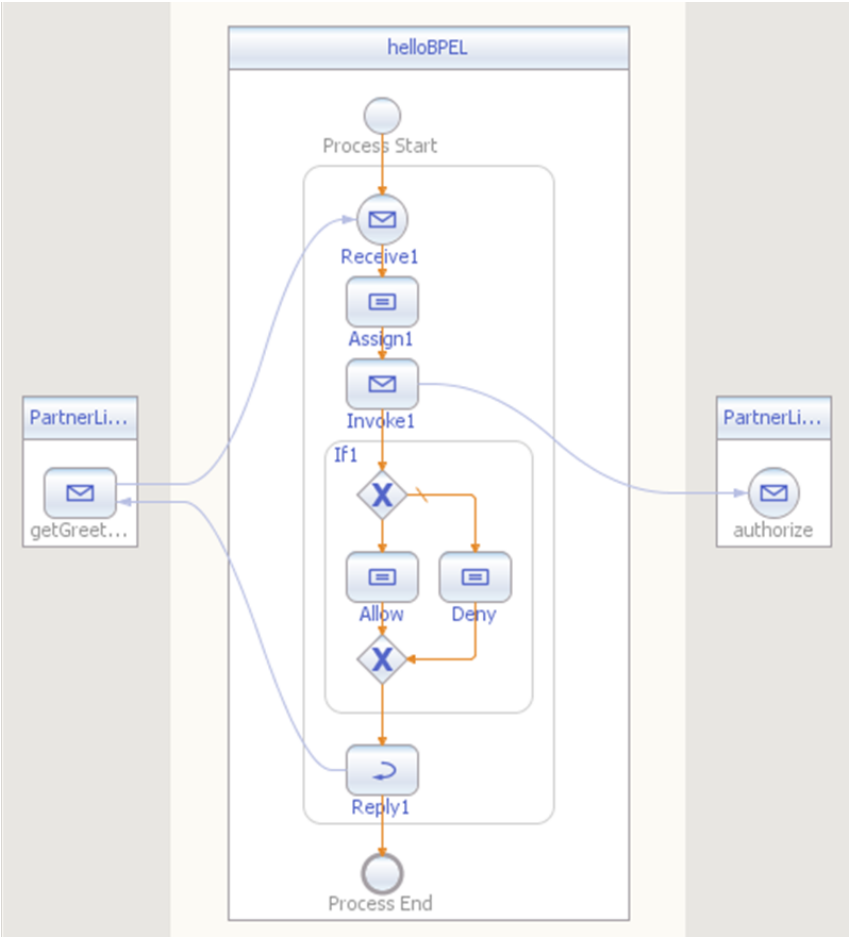


Рис. 64. Результат моделирования BPEL-процесса в среде NetBeans

5.7. Интеграция и развертывание бизнес-процесса

В рамках решения задачи интеграции бизнес-компонентов в сервис-орентированной архитектуре компанией Sun Microsystems была проведена работа над созданием спецификации, в деталях описывающей механизмы интеграции различных программных компонентов, построенных на принципах SOA. Результатом этой работы стала спецификация Java™ Business Integration (JBI) (JSR208 и JSR312). Она определяет открытый стандарт сборки, развертывания, исполнения, мониторинга комплексных приложений (composite applications), включающих в себя конфигурацию связей нескольких бизнес-компонентов.

Данную спецификацию поддерживают ряд крупных производителей программного обеспечения, в том числе с открытым исходным кодом.

Имплементацией JBI в рассматриваемом технологическом окружении является среда разработки NetBeans, позволяющая создавать, собирать, конфигурировать и разворачивать composite application, а также компонент OpenESB в составе сервера приложений Glassfish, отвечающий за среду исполнения JBI компонентов. Данные программные системы являются свободно распространяемыми и открытыми.

Рассмотрим последовательность шагов по сборке, конфигурированию и развертыванию комплексного приложения.

- Разработка комплексного приложения в среде разработки NetBeans начинается с создания проекта с именем HelloCA через меню File>New Project, где в категории SOA необходимо выбрать Composite Application.
- Затем следует добавить созданный ранее BPEL-процесс как JBI-компонент с помощью контекстного меню проекта Add JBI Module, где нужно указать путь до проекта HelloBPEL.
- После выполнения команды сборки проекта (пункт меню Run>Build Main Project) в окне редактора будет отображена схема связей бизнес-компонентов комплексного приложения, как представлено на рис. 65.

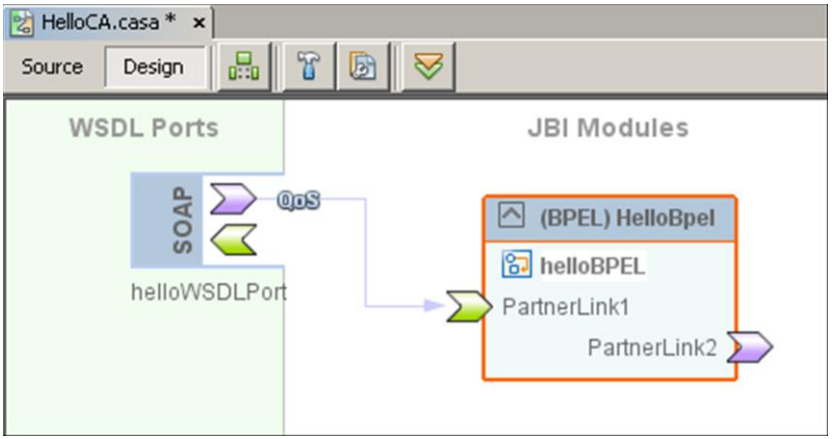


Рис. 65. Проект комплексного приложения до конфигурирования

Как видно из рис. 65, в конфигурации приложения не хватает привязки интерфейса AuthWS, к которому бизнес-процесс обращается через партнерскую связь PartnerLink2, к точке доступа, то есть к непосредственному сетевому адресу, на котором функционирует веб-сервис AuthWS. Чтобы привязать данный интерфейс к веб-сервису необходимо перетащить элемент SOAP WSDL Binding из палитры (Window>Palette) в поле WSDL Ports, соединить с ним PartnerLink2 и вызвать форму редактирования свойств через контекстное меню Properties добавленного SOAP порта.

Затем следует изменить свойства Endpoint Name на AuthWSPort и Location на адрес, где на предыдущих этапах разработки был развернут веб-сервис (см. рис. 66):

http://localhost:8080/AuthorizeWebServiceApp/AuthWSService

В результате перечисленных действий JBI компонент BPEL был сконфигурирован для работы с ранее развернутым веб-сервисом.

Развертывание сборки производится вызовом контекстного меню Deploy проекта composite application HelloCA.

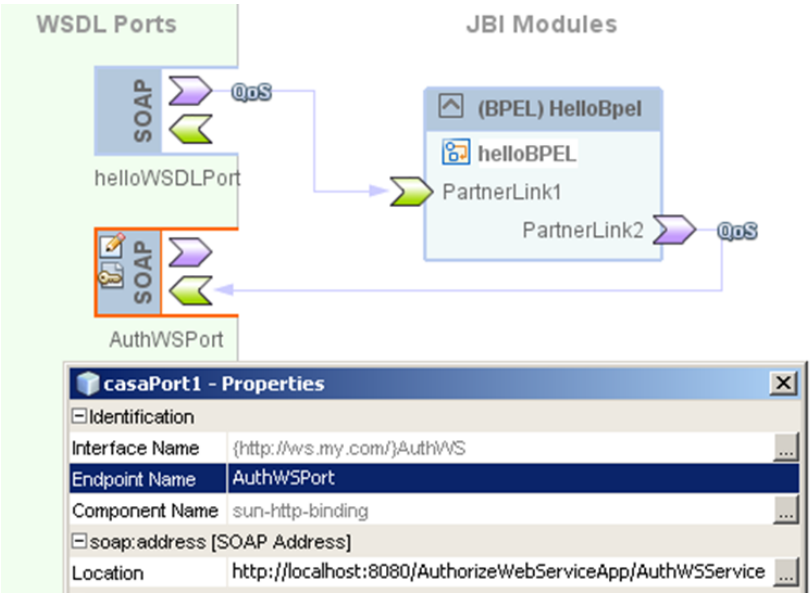


Рис. 66. Привязка интерфейса веб-сервиса к сетевому адресу точки доступа

Для целей тестирования необходимо узнать, по какому адресу будет зарегистрирован порт бизнес-процесса, принимающий запросы по протоколу HTTP. Для этого следует открыть форму со свойствами WSDL-порта с именем helloWSDLPort через контекстное меню Properties и скопировать значение поле Location:

http://localhost:${HttpDefaultPort}/helloWSDLService/helloWSDLPort

Во время развертывания сборки composite application среда исполнения подменяет строку ${HttpDefaultPort} на значение, присвоенное в конфигурации транспортного уровня сервера приложений (в рассматриваемом примере – 9080). Данное значение может быть выяснено непосредственно из NetBeans: необходимо открыть меню Window>Services, в дереве компонентов целевого сервера приложений Glassfish открыть свойства узла JBI>Binding Components>sun-http-binding, как изображено на рис. 67.

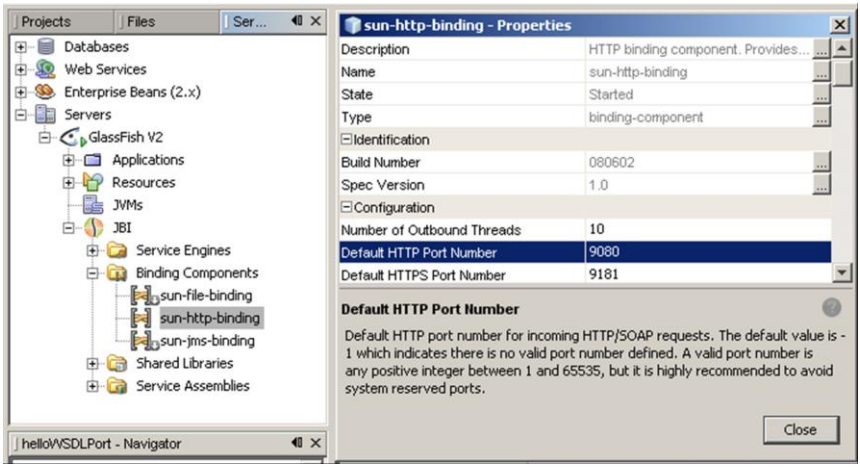


Рис. 67. Свойства транспортного уровня JBI-компонентов, разворачиваемых на сервере приложений Glassfish

Следует отметить, что среда исполнения BPEL позволяет проводить отладку бизнес-процесса в пошаговом режиме, используя как визуальное представление (граф BPEL-процесса), так и текстовое. Для включения режима пошаговой отладки необходимо изменить свойство JBI>Service Engine>sun-bpel-engine>Debug Enabled в настройках сервера приложений и начать сессию отладки через меню Debug>Attach Debugger>BPEL Debugger. Точку останова можно поставить на любом блоке BPEL-диаграммы через меню Debug>Toggle Line Breakpoint.

5.8. Тестирование программного комплекса

Согласно концепции SOA разработанный бизнес-процесс представлен внешнему окружению как веб-сервис с открытым интерфейсом и взаимодействующим по протоколу SOAP. Следовательно, функциональное тестирование программного комплекса, то есть совместное исполнение бизнес-процесса и сторонних вызываемых их сервисов, не отличается от процедуры, описанной в разделе 5.5.4, излагающим практическую сторону процесса тестирования веб-сервиса с помощью программного комплекса SoapUI.

Однако в понятие тестирования программного комплекса, как правило, включают дополнительные виды тестирования. Приведем

пример, как осуществить подготовку к нагрузочным испытаниям системы, используя описанные ранее инструменты.

Аналогичным образом, как это изложено в разделе 5.5.4, необходимо создать проект тестирования и указать URL-адрес WSDL-описания развернутого BPEL-процесса:

http://localhost:9080/helloWSDLService/helloWSDLPort?wsdl

Далее необходимо создать тестовый набор (test suite), тестовый сценарий (test case) и добавить в него шаг теста (SOAP Test Request), при необходимости добавить дополнительные проверки, например на наличие в SOAP-ответе строки "Приветствую Вас, Alice".

Затем, с помощью контекстного меню New LoadTest узла Load Tests сценария тестирования, добавляется новый нагрузочный тест. Задается конфигурация нагрузки: количество клиентских потоков, время проведения испытания, стратегия нагрузки и т.д. Разработанный программный комплекс способен держать нагрузку в 67 транзакций в секунду, как видно из рис. 68.

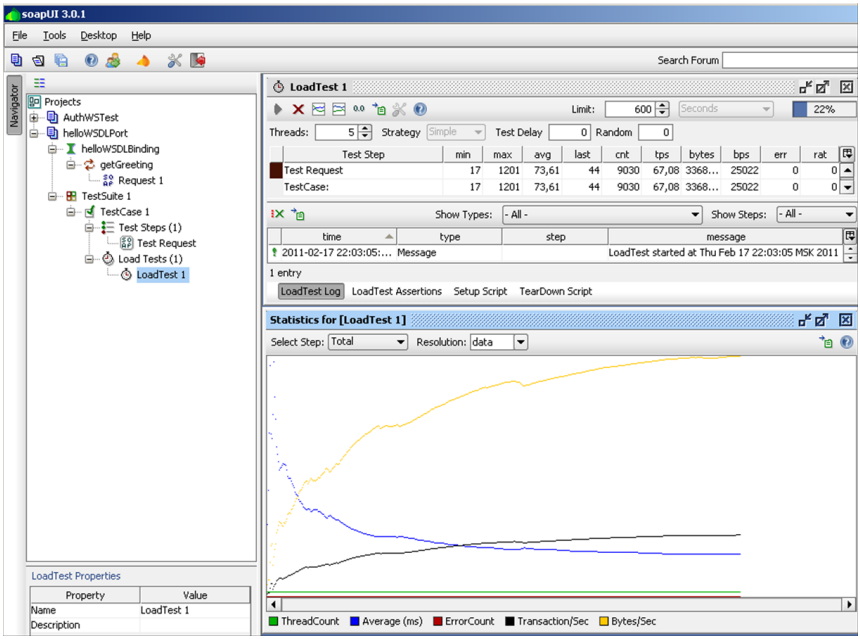


Рис. 68. Проведение нагрузочных испытаний BPEL-процесса

Содержание

Литература

1. F. Leymann, D. Roller, S. Thatte. Goals of the BPEL4WS Specification. http://www.oasis-open.org/committees/download.php/3249/Original%20Design%20Goals%20for%20the%20BPEL4WS%20Specification.doc)
2. C. Barreto, V. Bullard. Web Services Business Process Execution Language Version 2.0 Primer. 2007 http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.html
3. Business Process Model and Notation (BPMN) version 2.0. 2009 (http://www.omg.org/spec/BPMN/2.0)
4. T. Erl. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, 2005. 760 p.
5. CCITT. Languages and General Software Aspects for Telecommunication Systems. Concrete Graphical Syntax Summary (Z.100 Annex C1), 1988
6. Леоненков, А.В. Самоучитель UML 2. – СПб.: БХВ-Петербург, 2007. – 576 с.
7. Птицына, Л.К. Программное обеспечение компьютерных сетей. Моделирование механизмов синхронизации параллельных вычислительных процессов в системах мониторинга и управления: учеб. пособие. / Л.К. Птицына, Н.В. Соколова. – СПб.: Изд-во Политехн. ун-та, 2010. – 213 с.
8. Web Services Activity. http://www.w3.org/2002/ws/
9. D. Kulp. Web-Services Stack Comparison, 2010 http://wiki.apache.org/ws/StackComparison
10. A. Pelegri. Web Services Comparison Matrix Updated with GlassFish Implementation, 2006 http://blogs.sun.com/theaquarium/entry/web_services_comparison_matrix_updated

11. A. Pelegri. The New WebServices Stack in GlassFish V2, 2006
http://blogs.sun.com/theaquarium/entry/the_new_webservices_stack_in
12. E. Jendrock. The Java EE 6 Tutorial, 2010
http://download.oracle.com/javaee/6/tutorial/doc/
13. R. Grehan. Three open source Web service testing tools get high marks, 2007
http://www.infoworld.com/d/architecture/three-open-source-web-service-testing-tools-get-high-marks-995
14. R. Butek. Which style of WSDL should I use? 2005
http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/
15. Практическое руководство по языку XPath 2.0
http://www.w3schools.com/xpath/