УДК 602-507

ЯЗЫК МОДЕЛИРОВАНИЯ БИЗНЕС-ПРОЦЕССОВ BPDL

А. С. Толмачев,*
инженер-программист
ООО «Лектон»

Рассматривается новый язык описания бизнес-процессов BPDL, обладающий, в отличие от известных BPEL и XLANG, рядом важных свойств: простотой, гибкостью и расширяемостью.

Введение

Автоматизация бизнеса — далеко не новая для сферы информационных технологий тема. Говоря об автоматизации бизнеса, мы говорим об автоматизации бизнес-процессов организации. Бизнес-процесс — это последовательность операций, в ходе выполнения которых получается значимый для организации результат (продукты, услуги).

В последнее десятилетие были предприняты попытки найти общие языки для описания бизнес-процессов. Модель бизнес-процесса, описанная на таком языке, может быть затем реализована в соответствующей среде выполнения, как программа.

В этой статье рассматриваются бизнес-процессы, выполняющие интеграционную роль. Такие бизнес-процессы могут быть охарактеризованы следующим образом. Бизнес-процесс — это длящийся во времени вычислительный процесс, обладающий набором данных, которые могут создаваться, использоваться и уничтожаться в различные моменты времени, ориентированный на взаимодействие с внешними системами путем обмена XML-сообщениями.

Наиболее известными и распространенными языками моделирования бизнес-процессов такого рода на текущий момент являются языки BPEL4WS и XLANG. XLANG — это язык моделирования бизнес-процессов, разработанный Microsoft специально для интеграционной платформы Biztalk [1, 2]. Язык BPEL4WS (Business Process Execution Language For Web Services) — это язык, разработанный совместно IBM и Microsoft и имеющий открытый XML-синтаксис [3]. BPEL4WS и XLANG схожи по структуре и выразительной мощности. Для примера вкратце рассмотрим BPEL.

Язык BPEL опирается на использование веб-сервисов — одного из наиболее популярных подходов к решению интеграционных задач. Понятие веб-сервиса было разработано в целях стандартизации внешних интерфейсов взаимодействующих систем. Веб-сервисом может быть любая система, ориентированная на обмен сообщениями с внешними партнерами, снабженная опубликованным описанием на языке WSDL (Web Service Description Language) [4].

В BPEL любой бизнес-процесс описывается программой, состоящей из *деятельностей* (activities) — атомарных инструкций языка. Деятельности делятся на простые (primitive) и составные (structural). Простые деятельности служат для выполнения атомарных задач, таких как:

- отправка произвольного сообщения во внешнюю систему (деятельность send);
- ожидание сообщения из внешней системы (деятельность receive);
- операции над переменными бизнес-процесса (деятельность assign) и т. д.

Структурные деятельности позволяют объединять простые деятельности в алгоритмические конструкции:

- последовательность (sequence) группирует деятельности для выполнения друг за другом;
- поток (flow) служит для выполнения деятельностей параллельно;
- ветвление (switch) позволяет выполнять те или иные деятельности в зависимости от условия;
- цикл (while) служит для многократного выполнения последовательности деятельностей.

Кроме этого, BPEL имеет специальные конструкции для генерации исключений (throw) и их обработки (блоки faultHandlers, catch и catchAll). Также BPEL предусматривает очень важную для длительных транзакций (long-running transactions — LRT) возможность компенсации (блоки compensationHandlers и деятельность compensation). Длительные транзакции, в которых участвуют несколько сторон, невозможно проводить по принципам ACID (Atomicity, Consistence, Isolation, Durability), применяемым, в частности, в системах

* Научный руководитель — канд. физ.-мат. наук, доцент Санкт-Петербургского государственного политехнического университета Ф. А. Новиков.

управления базами данных. Поэтому в длительных транзакциях обычно используется принцип компенсации, в соответствии с которым для каждой операции определяется обратная ей.

К сожалению, BREL и XLANG имеют принципиальные ограничения и недостатки.

Основными из них являются следующие:

• отсутствие важных базовых функциональных конструкций (в BPEL, например, нельзя вызвать одну программу из другой в качестве подпрограммы);

• сложность языков (количество сущностей в каждом составляет более трех десятков);

• отсутствие механизмов расширения.

В течение последних лет были предприняты значительные усилия, направленные на то, чтобы сделать языки моделирования бизнес-процессов более доступными для неспециалистов в области программирования, в основном для бизнес-аналитиков. В этих целях OMG был принят стандарт BPMN (Business Process Modeling Notation) графической нотации для бизнес-процессов [5]. BPMN специфицирует набор графических элементов, описывает их семантику и определяет отображение своих диаграмм в BPEL. Семантическая модель BPMN более абстрактна, чем модель BPEL, она позволяет бизнес-аналитику описывать процесс, не задумываясь о деталях реализации. После трансляции в BPEL модель бизнес-процесса должна быть передана программисту, который необходимым образом уточнит ее и подготовит для выполнения.

Наличие перечисленных недостатков способствовало появлению идеи разработки нового простого расширяемого языка моделирования бизнес-процессов.

Описание языка BPDL

Предлагаемый язык BPDL (Business Process Description Language), в отличие от перечисленных языков, обладает следующими преимуществами:

• он расширяем — количество инструкций не ограничено;

• он ориентирован на предметную область — в язык включены инструкции, которые ориентированы на решение конкретных задач (выполнение платежа, получение баланса и т. д.).

Как и BPEL, BPDL имеет XML-синтаксис.

В BPDL любой бизнес-процесс обладает набором локальных данных (контекстом) и набором идентификаторов внешних систем, с которыми происходит обмен сообщениями.

Бизнес-процесс описывается программой. Атомарной инструкцией является шаг (step) — аналог деятельности в BPEL. Шаги группируются в задачи (tasks) — последовательности шагов с возможными ветвлениями. Набор связанных между собой задач, предназначенных для решения конкретной бизнес-задачи, составляет программу. Для любого шага может быть определен набор ветвей (branches) с условиями (conditions), наложенными на переменные контекста. Ветвь содержит последовательность шагов, которые выполняются, если выполняется условие. Шаги делятся на две основные категории:

• управляющие шаги, осуществляющие навигацию между задачами и управление выполнением бизнес-процесса;

• прикладные шаги, осуществляющие операции с сообщениями, локальными данными и т. д.

Определение 1. Переменной будем называть пару $v = (val, D)$, где $D = Type(v)$ — множество значений данной переменной; $val \in D$ — ее текущее значение.

Определение 2. Каналом будем называть пару $ch = (q, M)$, где q — это очередь, доступ к элементам которой организован по принципу FIFO, а $M = Type(ch)$ — множество сообщений — элементов, которые могут помещаться в очередь q.

Для канала определены следующие операции:

• $ch!m$ — операция отправки сообщения $m \in M$ в канал ch (сообщение помещается в конец очереди). Операцию отправки значения переменной v в канал ch ($Type(v) = M$) будем также обозначать $ch!v$;

• $ch?v$ — операция приема сообщения из канала ch (первый элемент извлекается из очереди и помещается в переменную v).

Операционная семантика BPDL определяется с помощью виртуальной машины BPM, выполняющей программы BPDL. Вычислительный процесс на BPM определяется набором $\{p, S, C, Ch\}$, где p — программа; S — стек программных вызовов; C — контекст, т. е. совокупность данных программы, представляющая собой набор переменных произвольных типов; Ch — набор доступных каналов передачи сообщений.

Программа p — это выражение на языке BPDL. Множество синтаксически правильных программ на BPDL будем обозначать P. Абстрактный синтаксис BPDL задается следующей грамматикой:

$$\text{Program} \rightarrow \text{Task\_ref} \quad (\text{Task})^+$$

$$\text{Task} \rightarrow \text{Sequence}$$

$$\text{Sequence} \rightarrow (\text{Step} \quad \text{Branch}^*)^+$$

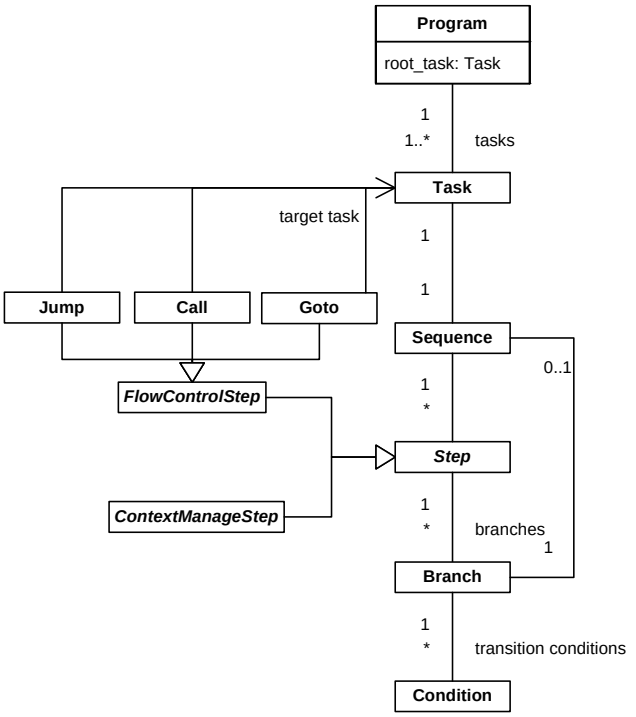
$$\text{Branch} \rightarrow (\text{Condition}^* \quad \text{Sequence}) \mid \text{Condition}^*$$

$$\text{Step} \rightarrow \text{ControlFlowStep} \mid \text{ContextManagementStep}$$

$$\text{ControlFlowStep} \rightarrow (\text{jump} \mid \text{goto} \mid \text{call})$$

$$\text{Task\_ref} \mid \text{return} \mid \text{finish}$$

Программа (Program) состоит из ссылки (Task_Ref) на корневую задачу и списка задач (Task). Задача содержит последовательность (Sequence). Каждая последовательность состоит из шагов (Step) с набором ветвей (Branch). Ветвь, в свою очередь, состоит из набора условий (Condition) ее выполнения и последовательности. Шаги делят-



■ *Рис. 1. Метамодель языка BPDL*

ся на управляющие (ControlFlowStep) и прикладные (ContextManageStep).

Управляющие шаги позволяют вызывать подзадачу (шаг call), перейти к ее выполнению с потерей точки вызова (шаг jump) и без его потери (шаг goto), осуществлять возврат из вызываемой подзадачи (шаг return), закончить выполнение бизнес-процесса (шаг finish).

Condition — это описание функции $C \to \{0, 1\}$. Прикладной шаг — это функция $C \times Q \to C$ (функция, выполняющая вычисление над переменными контекста с участием, возможно, обмена сообщениями по каналам из набора каналов Q).

BPDL включает следующие основные прикладные шаги:

- Service (посылает указанное сообщение в указанный канал и ожидает ответного сообщения);
- JavaScript (выполняет указанный код на JavaScript над переменными бизнес-процесса);
- XSLT (выполняет XSLT-преобразование над переменной).

Прочие шаги имеют ориентацию на предметную область: Payment выполняет платеж, Balance выполняет запрос баланса по карте и т. д.

Метамодель языка представлена на рис. 1.

Алгоритм интерпретации программы на BPDL

Текущая выполняемая инструкция в программе идентифицируется парой (task, step), где step — это выполняемый в данный момент шаг из задачи task. Стек S содержит пары (task, step). BPM, получив на вход программу p, выполняет ее, действуя по следующему алгоритму.

Алгоритм 1. Алгоритм интерпретации программы на BPDL.

```
Вход: p — программа, Q — набор каналов
Локальные переменные: C — контекст, S — стек
 1 root_task = p.root_task;
 2 (root_task, GetFirstStep(root_task)) -> S
 3 do
 4   (task, step) <- S
 5   if (type(step) = "jump") then
 6     S = ϕ {Очищаем стек}
 7     target_task = GetTargetTask(step)
 8     (target_task, GetFirstStep(target_task)) -> S
 9     continue
10   else if (type(step) = "call") then
11     (task, step) -> S { Сохраняем точку вызова }
12     target_task = GetTargetTask(step)
13     (target_task, GetFirstStep(target_task)) -> S
14     continue
15   else if (type(step) = "goto") then
16     target_task = GetTargetTask(step) {Переходим на
       целевое задание}
17     (target_task, GetFirstStep(target_task)) -> S
18     continue
19   else if (type(step) = "return") then
20     (task, step) <- S { Восстанавливаем точку вызова }
21   else if (type(step) = "finish") then
22     break { Завершаем вычислительный процесс }
23   else { Другой step }
24     ExecuteStep(step, C, Q);
25   end if
26
27   B = step.branches;
28   for b ∈ B do
29     Q = b.transition_conditions;
30     for q ∈ Q do
31       if (IsTrueCondition(q, C)) then
32         {условие выполняется — переходим на первый step
           в branch}
33         if (b.sequence != NULL) then
34           (task, b.sequence) -> S
35           continue
36         end if
37       end if
38     end for
39   end for
40   step = GetNextStep(task, step)
41   (task, step) -> S
42 end do
```

В алгоритме используются следующие функции:

GetFirstStep(t) — возвращает первый шаг в задаче t;

GetNextStep(t, s) — возвращает шаг, следующий за s в задаче t;

GetTargetTask(s) — возвращает задачу, на которую ссылается шаг s (подразумевается, что шаг имеет тип «jump», «goto» или «call»);

IsTrueCondition(q, C) — вычисляет истинность условия q над контекстом C;

ExecuteStep(s, C) — выполняет прикладной шаг.

Пример использования

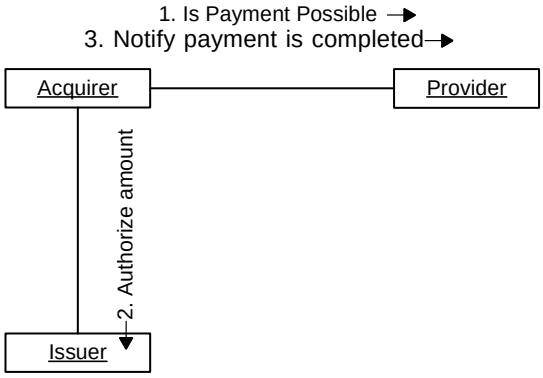
В качестве примера, иллюстрирующего применение BPDL, рассмотрим безналичный платеж в пользу поставщика услуг (таким поставщиком может быть мобильный оператор, оператор спут-

никового телевидения и т. д.). Платеж выполняется с терминала самообслуживания или банкомата. Терминал не обязательно принадлежит банку-эмитенту (выпустившему карту), а может принадлежать другому банку — члену платежной системы. Платеж выполняется в режиме реального времени, т. е. поставщик услуг сразу же узнает о выполнении платежа.

Обозначим роли, участвующие в операции:

- обслуживающий банк (acquirer) выполняет платеж в своей системе;
- банк-эмитент (issuer) содержит счет клиента;
- поставщик услуг (provider) принимает платеж.

Взаимодействие сторон при выполнении платежа представлено на рис. 2.



■ ***Рис. 2.*** *Порядок взаимодействия сторон при выполнении платежа*

Сначала проверяется возможность выполнения платежа со стороны поставщика услуг. В систему поставщика отправляется сообщение с запросом проверить возможность выполнения платежа (например, номер телефона, сумма платежа и его валюта) (шаг 1). Если ответ положителен, выполняется авторизация на стороне банка-эмитента (шаг 2). Эмитент проверяет, что в его системе заведена карта с таким номером и что на счету достаточно средств для выполнения платежа. Если проверки выполнены успешно, сумма платежа на клиентском счету блокируется и ответ следует положительный. Последним этапом выполняется уведомление поставщика о том, что в его пользу был совершен платеж (в системе банка, принимающего платеж, на счет поставщика была переведена указанная сумма).

Рассмотрим детально действия, которые выполняются в системе принимающего банка (рис. 3).

В первую очередь, платеж регистрируется в системе (Start operation). Совершаются необходимые проверки, создается платежный документ. В случае если поставщик отвергает платеж (шаг Request provider if payment is possible), операция завершается (End operation (failure)). Платежный документ при этом получает статус ошибки. То же самое происходит, если авторизация на стороне эмитента (Authorize amount) заканчивается неуспешно. В случае если ошибка происходит при уведомлении поставщика (Notify payment is completed), необходимы дополнительные действия по компенсации: в системе эмитента заблокированная сумма должна вернуться в распоряжение клиента (Rollback authorization).

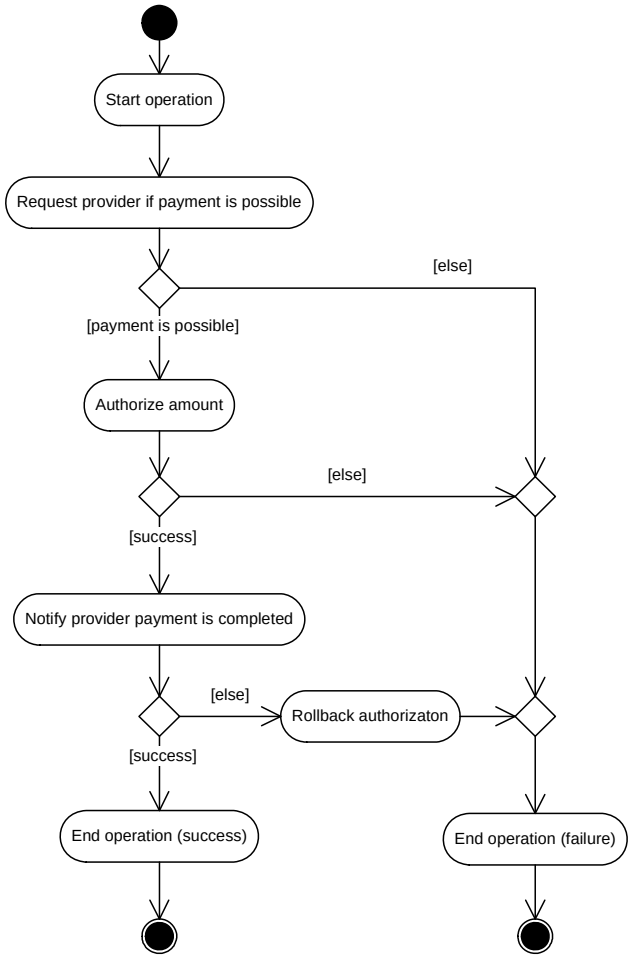
Если все шаги выполняются успешно, платежный документ получает положительный статус и операция завершается (End operation (success)).

Рассмотрим варианты реализации задачи на языке BPDL. Для этого создадим отдельную задачу (task). Будем считать, что следующие входные переменные существуют в контексте бизнес-процесса:

- device_contract: Contract;
- selected_contract: Contract;
- doc_msg: DocumentMsg.

Contract и DocumentMsg являются типами соответствующих XML-документов, содержащих информацию о контракте (счете, карте, терминале — другими словами, участнике транзакции) и о финансовом документе в общем виде.

Переменная device_contract содержит информацию о контракте терминала, selected_contract — о счете, с которого выполняется платеж, doc_msg представляет собой документ с указанными реквизитами.



■ ***Рис. 3.*** *Действия, выполняемые при совершении платежа*

Будем использовать следующие шаги:

• *StartOperation*. Шаг регистрирует финансовый документ в системе. Параметры:
  - doc_msg: *DocumentMsg* — финансовый документ с реквизитами;
  - device_contract: *Contract* — контракт терминала;
  - payer_contract: *Contract* — контракт счета-источника;
  - error_branch: *BranchRef* — ссылка на дочернюю ветвь-обработчик ошибки.

• *EndOperation*. Обновляет документ и присваивает ему статус (положительный или отрицательный). Параметры:
  - doc_msg: *DocumentMsg* — финансовый документ с реквизитами;
  - device_contract: *Contract* — контракт терминала;
  - payer_contract: *Contract* — контракт счета-источника;
  - error_branch: *BranchRef* — ссылка на дочернюю ветвь-обработчик ошибки.

• *Service*. Посылает сообщение во внешнюю систему и ожидает ответного сообщения. Параметры:
  - channel: *ChannelID* — идентификатор канала внешней системы;
  - template(...): *XML-Template* — шаблон сообщения, параметризуемый переменными (динамический XML);
  - error_branch: *BranchRef* — ссылка на дочернюю ветвь-обработчик ошибки;
  - timeout_branch: *BranchRef* — ссылка на дочернюю ветвь-обработчик.

В следующем BPDL-псевдокоде используются обозначения:

• PROVIDER — идентификатор канала системы поставщика;

• ISSUER — идентификатор канала системы банка-эмитента;

• PROVIDER_CHECK — шаблон XML-запроса проверки возможности платежа;

• AUTH — шаблон авторизационного XML-запроса;

• PROVIDER_NOTIFY — шаблон уведомления о выполненном платеже;

• REVERSE — шаблон запроса на откат операции.

Алгоритм 2. BPDL-псевдокод программы, реализующей платеж.

```
Вход:
 device_contract: Contract;
 selected_contract: Contract;
 doc_msg: DocumentMsg;
Выход:
 doc_msg: DocumentMsg;

Step [ StartOperation(doc_msg, device_contract, selected_contract,
                  error_branch) ]
Step [ Service(PROVIDER, PROVIDER_CHECK(doc_msg, device_contract,
            selected_contract), error_branch, error_branch) ]
 Branch error_branch [ { шаг закончился неуспешно }
 Condition [ doc_msg.RespCode != 0 ]
    Step [ EndOperation(doc_msg, device_contract,
     selected_contract,
                        error_branch) ]
     Step [ Return ]
]
Step [ Service(ISSUER, AUTH(doc_msg, device_contract,
              selected_contract), error_branch, error_branch) ]
 Branch error_branch [ { шаг закончился неуспешно }
 Condition [ doc_msg.RespCode != 0 ]
    Step [ EndOperation(doc_msg, device_contract,
     selected_contract,
                        error_branch) ]
     Step [ Return ]
]
Step [ Service(PROVIDER, PROVIDER_NOTIFY(doc_msg, device_contract,
            selected_contract), error_branch, error_branch) ]
 Branch error_branch [ { шаг закончился неуспешно }
 Condition [ doc_msg.RespCode != 0 ]
    Step [ Service(ISSUER, REVERSE(doc_msg,
     device_contract,
                    selected_contract), error_branch,
                    error_branch ]
      Branch error_branch [ { шаг закончился неуспешно }
        Step [ Return ]
      ]
    Step [ EndOperation(doc_msg, device_contract,
     selected_contract,
                        error_branch) ]
      Branch error_branch [ { шаг закончился неуспешно }
        Step [ Return ]
      ]
   Step [ Return ]
]
```

Вторым способом решения задачи является использование шага «Платеж»:

• Payment — выполняет платеж по Алгоритму 2. Параметры:
  - doc_msg: *DocumentMsg* — финансовый документ с реквизитами;
  - device_contract: *Contract* — контракт терминала;
  - payer_contract: *Contract* — контракт счета-источника.

Для чего необходимо реализовывать Алгоритм 2 в виде отдельного шага? Для этого есть следующие причины.

• Ориентация на прикладную область. Алгоритм 2 является упрощенным: в реальности логика выполнения операций более сложна и содержит больше деталей. Реализация такого алгоритма на BPDL потребовала бы большого количества кода на JavaScript и низкоуровневых шагов (например, шага «Вызов произвольной хранимой процедуры»), чего хотелось бы избежать.

• Удобный графический интерфейс. Удобство использования шагов во многом зависит от пользовательского интерфейса. Добавляя шаг в программу, пользователь указывает его входные и выходные параметры, причем зачастую они имеют структуру более сложную, нежели линейная, как последовательность аргументов функции. Проблема создания пользовательского интерфейса шага лежит на программистах. Задача в этом смысле не может заменить шаг, поскольку написанная пользователем, она будет вызвана через общий для всех задач интерфейс шага call.

• Производительность. Будучи реализованным как отдельный шаг на Java, алгоритм будет выполняться быстрее, чем реализованный задачей на BPDL.

Рассмотренный пример проиллюстрировал применение BPDL для программирования распределенных финансовых транзакций в среде, ориентированной на обмен сообщениями. Другим примером использования языка может служить программирование приложения информационного банковского киоска или банкомата. В этом случае используются прикладные шаги, отвечающие за посылку на терминал специализированной команды (ожидать ввода карты, показать экран, принять купюры, распечатать чек) и обработку результатов ее выполнения. Вместе с набором шагов для программирования финансовых транзакций эти шаги предоставляют удобную базу для реализации разнообразных приложений для каналов банковского самообслуживания. Ориентация языка на решение конкретной задачи в таком случае делает процесс программирования простым и доступным не только программистам, но и сотрудникам внедрения, и служащим IT-подразделений банка.

Заключение

Мы рассмотрели новый язык описания бизнес-процессов BPDL. Предлагаемый язык, в отличие от известных BPEL и XLANG, обладает рядом преимуществ: он прост, гибок и расширяем. Расширяемость позволяет пополнять язык инструкциями, ориентированными на конкретную предметную область, тем самым увеличивая круг его потенциальных пользователей.

Свойства языка проиллюстрированы взятым из практики примером.

Литература

1. **Woodgate S.** Microsoft BizTalk Server 2004 Unleashed. Sams Publishing, 2004.
2. **Chappell D.** Enterprise Service Bus. O'Reilly, 2004. P. 247.
3. **Matjaz B. J.** Business Process Execution Language for Web Services. Packt Puplishing, 2004. P. 372.
4. **Ньюкомер Э.** Веб-сервисы для профессионалов. СПб.: Питер, 2003. 256 с.
5. Business Process Modeling Notation Specification. www.omg.org.