Введение

Корпоративные приложения не могут существовать обособленно одно от другого. Программа, установленная в магазине розничной продажи, не будет эффективной без взаимодействия с программой, установленной на складе, а календарь на КПК — без синхронизации с сервером расписаний.

Как правило, разработчики интеграционных решений сталкиваются со следующими вызовами.

- **Ненадежность сети передачи данных.** Все интеграционные решения предполагают передачу информации между устройствами. В отличие от процессов, выполняющихся в пределах одного компьютера, распределенной вычислительной среде присущ целый ряд недостатков. Зачастую общающиеся системы разделены континентами, что вынуждает передавать данные по телефонным линиям, сегментам локальных сетей, через маршрутизаторы, коммутаторы, общедоступные сети и спутниковые каналы связи. Доставка информации на каждом из этих этапов связана с задержкой и риском потери.
- **Низкая скорость передачи данных.** Время доставки данных через компьютерную сеть на порядок больше времени вызова локального метода. Таким образом, создание распределенного приложения требует применения иных принципов проектирования, чем создание приложения, выполняющегося в пределах одного компьютера.
- **Различия между приложениями.** Интеграционное решение должно учитывать все различия (язык программирования, платформа, формат данных), существующие между объединяемыми системами.
- **Неизбежность изменений.** Интеграционное решение должно иметь возможность адаптации к изменению объединяемых им приложений. Зачастую преобразования в одной системе влекут за собой непредсказуемые последствия для других систем. Поэтому при интеграции приложений важно уменьшить их взаимозависимость за счет так называемого *слабого связывания*.

Для преодоления описанных выше трудностей можно воспользоваться четырьмя основными подходами.

1. ***Передача файла* (*File Transfer*, с. 80).** Одно приложение создает файл, а другое приложение считывает его. Приложения должны согласовать имя файла, его расположение, формат, время записи/считывания, а также процедуру удаления.
2. ***Общая база данных* (*Shared Database*, с. 83).** Несколько приложений используют общую логическую структуру данных, которой соответствует одна физическая база данных. Наличие единого хранилища данных устраняет проблему передачи информации между приложениями.

3. ***Удаленный вызов процедуры* (*Remote Procedure Invocation*, с. 85).** Приложение предоставляет доступ к части своей функциональности посредством удаленного вызова процедуры. Взаимодействие между приложениями осуществляется синхронно в режиме реального времени.
4. ***Обмен сообщениями* (*Messaging*, с. 87).** Приложение размещает сообщение в общем канале, которое затем считывается другим приложением. Приложения должны согласовать канал, а также формат сообщения. Взаимодействие между приложениями осуществляется в асинхронном режиме.

Каждый из предложенных подходов имеет собственные преимущества и недостатки. На практике приложения могут быть интегрированы несколькими способами таким образом, чтобы использовать только сильные стороны того или иного подхода.

Что такое обмен сообщениями

Эта книга посвящена интеграции приложений с помощью обмена сообщениями. Чтобы лучше понять смысл обмена сообщениями, рассмотрим систему телефонной связи. Телефонный разговор является ярким примером синхронного взаимодействия. Абонент может начать общение с вызываемой стороной только в том случае, если последняя окажется свободной в момент звонка. Привнесение в эту систему автоответчика делает ее асинхронной. Если абонент не отвечает, ему можно оставить голосовое сообщение, которое он сможет прослушать в удобное для него время. Это намного проще, чем пытаться дозвониться до абонента вновь и вновь. “Сохранение” части телефонного разговора в виде сообщения и помещение его в очередь для последующего прослушивания наглядно иллюстрирует сущность обмена сообщениями.

Обмен сообщениями — это технология высокоскоростного асинхронного взаимодействия между программами с гарантией доставки информации. Программы взаимодействуют между собой, обмениваясь пакетами данных, называемыми *сообщениями*. *Канал*, или *очередь*, — это логический маршрут, объединяющий программы и использующийся для транспортировки сообщений. Канал напоминает массив сообщений, доступный для одновременного использования многими приложениями. *Отправитель*, или *поставщик*, — это программа, отправляющая сообщение путем его размещения в канале. *Получатель*, или *потребитель*, — это программа, получающая (а затем удаляющая) сообщение путем его считывания из канала.

Сообщение представляет собой некоторую структуру данных — строку, байтовый массив, запись или объект. Оно может быть интерпретировано непосредственно как содержащиеся в нем данные, как команда, которую необходимо выполнить получателю, или как описание события, произошедшего на стороне отправителя. Сообщение состоит из двух частей — заголовка и тела. *Заголовок* сообщения содержит метаданные (кто отправил сообщение, куда его следует передать и т.п.), которые используются системой обмена сообщениями и игнорируются получателем сообщения. *Тело* сообщения содержит полезную информацию, которая, как правило, игнорируется системой обмена сообщениями. Упоминая сообщение в разговоре, разработчик приложения обычно имеет в виду информацию, содержащуюся в теле сообщения.

По сравнению с тремя оставшимися способами интеграции приложений, опыт работы с системами обмена сообщениями имеет весьма ограниченное число разработчиков. Как следствие применение архитектуры асинхронного обмена сообщениями зачастую требует переосмыслить подход к созданию приложений.

Что такое система обмена сообщениями

Функциональная часть обмена сообщениями обеспечивается отдельной программной системой, называемой *системой обмена сообщениями* или *связующим ПО, ориентированным на обмен сообщениями* (*message-oriented middleware — MOM*). Система обмена сообщениями имеет много общего с системой баз данных. В частности, схеме базы данных, использующейся для определения формата хранимой информации, соответствуют каналы обмена сообщениями. Основная задача системы баз данных — обеспечить надежное хранение информации, а основная задача системы обмена сообщениями — гарантировать доставку сообщений с компьютера отправителя на компьютер получателя.

Необходимость наличия системы обмена сообщениями обусловливается ненадежностью сетей передачи данных. Сбой в компьютерной сети — одна из самых распространенных причин неудавшейся доставки сообщения от отправителя к получателю. Система обмена сообщениями позволяет гарантировать доставку информации за счет повторной отправки сообщения (до тех пор, пока оно не будет принято получателем). В идеальных условиях сообщение доставляется с первого раза, однако, к сожалению, так бывает далеко не всегда.

Процедура передачи сообщения от отправителя к получателю состоит из пяти основных этапов.

1. **Создание.** Отправитель создает сообщение, содержащее полезную информацию.
2. **Отправка.** Отправитель помещает сообщение в канал.
3. **Доставка.** Система обмена сообщениями доставляет сообщение с компьютера отправителя на компьютер получателя.
4. **Получение.** Получатель извлекает сообщение из канала.
5. **Обработка.** Получатель считывает полезную информацию.

На рис. 1 проиллюстрированы все основные этапы процедуры передачи сообщения.

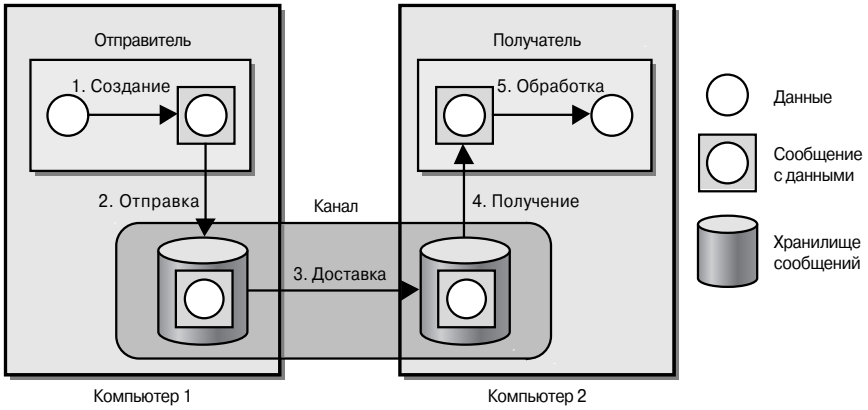

Рис. 1. Основные этапы передачи сообщения от отправителя к получателю с использованием системы обмена сообщениями

На рис. 1 также отражены две важные концепции обмена сообщениями.

1. **"Отправить и забыть".** Поместив сообщение в канал на этапе 2, отправитель может не заботиться о его дальнейшей судьбе — гарантированную доставку сообщения получателю обеспечивает система обмена сообщениями.
2. **Передача с промежуточным хранением.** На этапе 2 система обмена сообщениями сохраняет сообщение, помещенное отправителем в канал, на компьютере отправителя (в оперативной памяти или на жестком диске). На этапе 3 система обмена сообщениями доставляет сообщение с компьютера отправителя на компьютер получателя и сохраняет его на компьютере получателя. Процесс передачи сообщения с промежуточным хранением повторяется, если для достижения компьютера получателя сообщение должно пройти через несколько других компьютеров.

В описанной выше схеме этапы создания, отправки, получения и обработки сообщения могут показаться излишними. Действительно, почему бы просто не передать нужную информацию от отправителя к получателю? Создание сообщения и его передача системе обмена сообщениями позволяет делегировать последней всю ответственность за доставку данных, тем самым обеспечив надежную передачу единственной копии полезной информации получателю.

Преимущества обмена сообщениями

Обмен сообщениями обладает множеством преимуществ по сравнению с другими технологиями интеграции приложений. Коротко говоря, обмен сообщениями более быстр, чем *передача файла* (*File Transfer*, с. 80), обладает лучшей инкапсуляцией по сравнению с *общей базой данных* (*Shared Database*, с. 83) и более надежен, чем *удаленный вызов процедуры* (*Remote Procedure Invocation*, с. 85).

Ниже перечислены дополнительные преимущества, которые позволяет получить технология обмена сообщениями.

- **Удаленное взаимодействие.** Обмен сообщениями позволяет наладить взаимодействие между отдельными приложениями. Два объекта, относящихся к одному и тому же процессу, могут совместно использовать данные, размещенные в оперативной памяти. Передача информации с одного компьютера на другой требует выполнения "сериализации" данных — преобразования соответствующих объектов в байтовый поток с целью последующей отправки по сети. Как было сказано выше, приложение может делегировать всю ответственность за передачу данных системе обмена сообщениями, тем самым избавляясь от части сложной функциональности.
- **Платформенная/языковая интеграция.** Зачастую удаленные системы создаются с использованием различных платформ, технологий и языков программирования. Интеграция разнородных систем требует использования связующего ПО, в качестве которого может выступить система обмена сообщениями. Идея использования системы обмена сообщениями в качестве универсального связующего звена между приложениями была положена в основу шаблона *шина сообщений* (*Message Bus*, с. 162).
- **Асинхронное взаимодействие.** Обмен сообщениями позволяет наладить взаимодействие между приложениями по принципу *отправил и забыл* (*send-and-forget*). В соответствии с этим принципом отправитель не обязан ожидать подтверждение о получении и обработке сообщения от принимающей стороны; более того, он также не обязан ожидать подтверждение о доставке сообщения от системы обмена сообщениями. Единственное, о чем следует позаботиться отправителю, — это дождаться подтверждения об отправке сообщения, т.е. о его помещении в канал. Как только сообщение будет передано системе обмена сообщениями, отправитель может приступить к выполнению имеющихся у него задач.
- **Рассогласование во времени.** При синхронном взаимодействии отправитель должен дождаться завершения обработки вызова получателем прежде, чем сделать новый вызов. Таким образом, скорость размещения вызовов отправителем ограничена скоростью их обработки получателем. Асинхронное взаимодействие позволяет размещать и обрабатывать вызовы с разной скоростью, что существенно повышает эффективность взаимодействия между приложениями.
- **Регулирование нагрузки.** Слишком большое число удаленных вызовов процедур за короткий промежуток времени может привести к перегрузке получателя, снижению его производительности и даже выходу из строя. Система обмена сообщениями формирует очередь запросов, позволяя получателю контролировать скорость их обработки. Поскольку взаимодействие осуществляется в асинхронном режиме, регулирование нагрузки на стороне получателя не оказывает негативного влияния на отправителя.
- **Надежное взаимодействие.** В отличие от удаленного вызова процедуры, обмен сообщениями позволяет наладить надежное взаимодействие между приложениями за счет подхода, получившего название *передача с промежуточным хранением* (*store-and-forward*). Как упоминалось выше, сообщение представляет собой единицу передачи информации, инкапсулирующую полезные данные. Когда отправитель по-

мещает сообщение в канал, система обмена сообщениями сохраняет его на компьютере отправителя. Затем сообщение доставляется получателю и сохраняется на его компьютере. Предположим, что сохранение сообщения на компьютерах отправителя и получателя является надежной операцией. (Чтобы сделать ее еще надежнее, сообщение можно сохранять не в памяти, а на диске компьютера, как описывается шаблоном *гарантированная доставка* (*Guaranteed Delivery*, с. 149).) Единственным слабым звеном передачи с промежуточным хранением является доставка сообщения на компьютер получателя. Чтобы нивелировать негативное влияние возможных сбоев при передаче сообщения, система обмена сообщениями пересылает его до тех пор, пока оно не будет доставлено по назначению. Автоматическая пересылка сообщения позволяет исключить риск потери информации при возникновении сбоя в сети или на компьютере получателя, что, в свою очередь, делает возможным применение принципа взаимодействия между приложениями "отправил и забыл".

- **Работа без подключения к сети.** Некоторые приложения ориентированы на работу без подключения к сети. Как правило, подобные приложения предназначены для выполнения на ноутбуках или КПК и периодически (при наличии сетевого подключения) синхронизируют данные с сервером. Обмен сообщениями — идеальное решение для синхронизации, позволяющее накапливать данные в очереди до тех пор, пока приложение не получит доступ к сети.
- **Посредничество.** Система обмена сообщениями выступает в роли *посредника* (*Mediator*) [12] между взаимодействующими приложениями. Приложение может использовать систему обмена сообщениями в качестве каталога доступных для интеграции приложений и служб. Если приложение потеряет связь с другими приложениями, ему понадобится восстановить соединение только с системой обмена сообщениями, а не с каждым отдельным приложением. Функция посредника может потребовать от системы обмена сообщениями высокой доступности, балансировки нагрузки, устойчивости к отказам сетевых соединений, а также поддержки качества обслуживания.
- **Управление потоками.** Асинхронное взаимодействие позволяет приложению не ожидать результата выполнения задачи другим приложением. Вместо этого приложение может воспользоваться обратным вызовом, уведомляющим его о поступлении ответа (шаблон *запрос-ответ — Request-Reply*, с. 177). Большое число заблокированных потоков, а также потоки, заблокированные в течение длительного времени, могут оказать негативное воздействие на работу приложения. Кроме того, такие потоки трудно восстановить в случае сбоя приложения и его последующего перезапуска. Использование обратного вызова позволяет минимизировать количество заблокированных потоков, обеспечить стабильную работу приложения и определить потоки, которые должны быть восстановлены при его перезапуске.

Итак, существует несколько причин, свидетельствующих в пользу обмена сообщениями. Некоторые из них носят сугубо технический характер, в то время как остальные представляют собой стратегические решения, принимающиеся на этапе проектирования приложения. Безусловно, каждое из вышеперечисленных преимуществ обмена сообще-

ниями будет иметь различный вес в контексте конкретных требований, предъявляемых к приложению. Однако мы уверены, что каждое из них является достаточным аргументом для использования технологии обмена сообщениями при интеграции приложений.

Недостатки асинхронного обмена сообщениями

Несмотря на то что технология асинхронного обмена сообщениями позволяет преодолеть множество трудностей, связанных с интеграцией разнородных приложений, она не лишена недостатков. Некоторые из них являются неотъемлемой частью асинхронной модели взаимодействия, в то время как остальные зависят от конкретной реализации системы обмена сообщениями.

- **Сложная модель программирования.** Асинхронный обмен сообщениями требует от разработчиков использования модели событийно управляемого программирования. В этом случае логика приложения разбивается на множество обработчиков событий, реагирующих на входящие сообщения. Подобную систему гораздо труднее программировать и отлаживать, чем систему, основанную на вызове методов. Так, эквивалентом простого вызова метода в модели событийно управляемого программирования является совокупность, состоящая из сообщения с запросом, канала запроса, сообщения с ответом, канала ответа, идентификатора корреляции и очереди сообщений недопустимого формата (см. шаблон *запрос-ответ — Request-Reply*, с. 177).
- **Порядок доставки сообщений.** Система обмена сообщениями гарантирует доставку сообщения от отправителя к получателю, не оговаривая требующееся для этого время. В результате может быть нарушен порядок, в котором были отправлены сообщения. Если последовательность доставки сообщений имеет значение, ее нужно восстановить, как описано в шаблоне *преобразователь порядка* (*Resequencer*, с. 297).
- **Необходимость реализации синхронной модели.** Не все приложения могут взаимодействовать по принципу "отправил и забыл". К примеру, пользовательский запрос о наличии авиабилетов должен быть обработан немедленно, а не в течение неопределенного промежутка времени. Следовательно, в некоторых системах обмена сообщениями должен быть предусмотрен баланс между синхронной и асинхронной моделями взаимодействия.
- **Производительность.** Системы обмена сообщениями вносят дополнительные издержки в процесс взаимодействия между приложениями. На создание, отправку, получение и обработку сообщения уходят время и ресурсы. К тому же передача большого объема данных может повлечь за собой создание несметного числа сообщений. Так, зачастую интеграция двух существующих систем начинается с репликации всех необходимых данных. Репликация большого объема информации с помощью средств ETL (Extract, Transform and Load — "извлечение, преобразование, загрузка") гораздо эффективнее репликации с помощью обмена сообщениями. Таким образом, обмен сообщениями рекомендуется применять для синхронизации данных между приложениями, а не для их первичной репликации.
- **Ограниченная поддержка программными платформами.** Многие коммерческие системы обмена сообщениями недоступны для всех платформ. Зачастую единственный способ интеграции приложений заключается в использовании протокола

FTP, поскольку целевая программная платформа не поддерживается конкретной системой обмена сообщениями.

- **Зависимость от компании-разработчика.** Коммерческие системы обмена сообщениями могут основываться на использовании закрытых протоколов. Даже такая общепринятая спецификация обмена сообщениями, как JMS, не определяет подробностей реализации конкретного решения. В результате различные системы обмена сообщениями оказываются неспособными к взаимодействию друг с другом. Это может привести к возникновению новой задачи интеграции: интеграции нескольких различных интеграционных решений! (См. шаблон *мост обмена сообщениями Messaging Bridge*, с. 159).

Подведем итог. Технология асинхронного обмена сообщениями не решает всех задач интеграции. Более того, ее использование может привести к необходимости преодоления новых трудностей. Взвесьте все "за" и "против", прежде чем приступить к реализации конкретного интеграционного решения с помощью обмена сообщениями.

Мыслим асинхронно

Обмен сообщениями — это технология асинхронного взаимодействия приложений с гарантией доставки данных. В то же время большинство приложений использует синхронные вызовы функций; например, процедура вызывает подпроцедуру, один метод вызывает другой или же процедура вызывает другую процедуру через технологию удаленного вызова (такую, как CORBA или DCOM). Синхронный вызов требует от вызывающего процесса ожидания завершения выполнения функции подпроцессом. Даже при использовании удаленного вызова, подразумевающего выполнение подпроцедуры в отдельном процессе, вызывающий процесс приостанавливает свое выполнение до возврата управления (а также результата выполнения подпроцедуры). При использовании асинхронного обмена сообщениями приложения могут взаимодействовать по принципу "отправил и забыл", позволяющему вызывающему приложению разместить сообщение в канале и тотчас же вернуться к выполнению текущей задачи. Другими словами, вызывающая процедура продолжает свое выполнение во время вызова подпроцедуры, как показано на рис. 2.

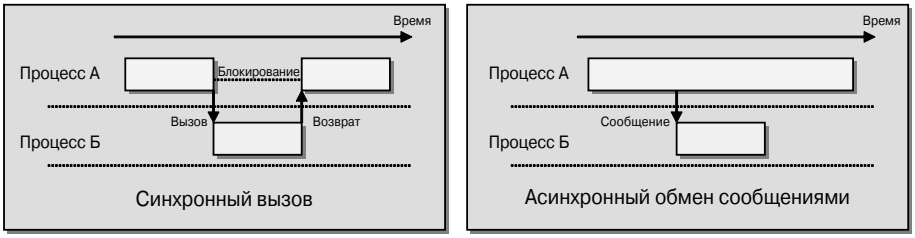

Рис. 2. Семантика синхронного и асинхронного вызовов

Асинхронное взаимодействие имеет ряд отличительных особенностей. Во-первых, речь идет о более чем одном потоке выполнения. Наличие нескольких потоков позволяет подпроцедурам выполняться одновременно, что существенно увеличивает производительность приложения, однако затрудняет его отладку. Во-вторых, результат выполнения подпроцедуры (если таковой имеется) возвращается посредством механизма обратного вызова. С одной стороны, это позволяет увеличить производительность, так как вызывающий процесс может заняться выполнением других задач, не дожидаясь возвращения результата. С другой стороны, вызывающий процесс должен быть готов к обработке результата в любой момент (даже во время выполнения другой задачи), а также помнить контекст, в котором был осуществлен вызов. В-третьих, асинхронные подпроцессы могут выполняться в любом порядке. Это накладывает на вызывающий процесс еще одно требование: уметь обрабатывать полученные результаты с учетом их источника и времени получения. Таким образом, асинхронная модель взаимодействия имеет несколько неоспоримых преимуществ, однако требует от разработчика переосмыслить способ использования процедурой своих подпроцедур.

Распределенное приложение или интеграция приложений

Зачастую корпоративные приложения распределяются между несколькими компьютерами за счет использования *n*-уровневой архитектуры (усовершенствованный вариант архитектуры клиент/сервер). Несмотря на то что процессы *n*-уровневого приложения выполняются на нескольких компьютерах и взаимодействуют между собой, интеграция приложений существенно отличается от распределенного приложения.

Почему же использование *n*-уровневой архитектуры не позволяет говорить об интеграции? Во-первых, взаимодействие между общающимися сторонами построено по принципу сильной связи — ни один из уровней приложения не может функционировать без оставшихся уровней. Во-вторых, взаимодействие между уровнями синхронно. В-третьих, пользователи приложения (одно- или многоуровневого) — это люди, которые привыкли к быстрому отклику системы.

В то же время интеграция подразумевает налаживание взаимодействия между независящими друг от друга приложениями по принципу слабой связи. Каждое из интегрированных приложений выполняет определенный круг задач, обращаясь к другим приложениям для получения некоторой дополнительной функциональности. Интегрированные приложения взаимодействуют асинхронно, что позволяет им продолжать работу, не дожидаясь ответа от вызываемой стороны. Это же делает возможным отказ от жестких временных ограничений, присущих пользовательским приложениям.

Коммерческие системы обмена сообщениями

Очевидные преимущества интеграционных решений, использующих асинхронный обмен сообщениями, дали толчок к созданию рынка соответствующего связующего ПО и средств разработки. Ниже перечислены четыре основные категории коммерческих продуктов, ориентированных на обмен сообщениями.

1. **Операционные системы.** Популярность технологии обмена сообщениями побудила разработчиков программного обеспечения интегрировать необходимую инфраструктуру в операционные системы и СУБД. К примеру, корпорация Microsoft включила в состав операционных систем Windows 2000 и Windows XP службу Microsoft Message Queuing (MSMQ). Служба MSMQ доступна посредством нескольких API, включая компоненты COM и классы пространства имен Microsoft .NET `System.Messaging`. Аналогичная функциональность (Oracle AQ) была реализована в СУБД Oracle.
2. **Серверы приложений.** Впервые компания Sun Microsystems включила спецификацию Java Messaging Service (JMS) в версию 1.2 платформы J2EE. С тех пор практически все серверы приложений J2EE (такие, как IBM WebSphere и BEA WebLogic) включают в себя реализацию JMS. Кроме того, эталонная реализация JMS поставляется в составе J2EE JDK.
3. **Решения для интеграции корпоративных приложений.** Как правило, решения для интеграции корпоративных приложений обладают богатой функциональностью и включают в себя систему обмена сообщениями, средства автоматизации документооборота и деловых операций, средства создания порталов и др. Наиболее известными EAI-решениями являются IBM WebSphere MQ, Microsoft BizTalk, TIBCO, WebMethods, SeeBeyond, Vitria, CrossWorlds и др. Некоторые из этих продуктов содержат несколько различных API, обеспечивающих обмен сообщениями, в то время как остальные (например, SonicSoftware и Fiorano) ориентируются исключительно на JMS.
4. **Средства создания Web-служб.** Web-службы привлекли к себе огромный интерес со стороны разработчиков интеграционных решений. В настоящее время усилия комитетов и консорциумов по стандартизации сосредоточены на принятии спецификации технологии надежного обмена сообщениями на основе Web-служб (стандарты WS-Reliability, WS-ReliableMessaging и ebMS). В то же время на рынке появляется все больше и больше продуктов, реализующих маршрутизацию, преобразование и управление решениями, основанными на использовании Web-служб.

Шаблоны, представленные в этой книге, не ориентированы на конкретного поставщика ПО и применимы к большинству интеграционных решений, основанных на обмене сообщениями. К сожалению, практически все существующие на рынке системы обмена сообщениями используют собственную терминологию. Мы же постарались подобрать простые, описательные имена шаблонов, не зависящие от конкретной технологии или решения.

Многие поставщики интеграционных решений, основанных на обмене сообщениями, реализовали некоторые из представленных в этой книге шаблонов в своих продуктах. Читатели, знакомые с терминологией конкретного поставщика, не должны испытывать трудностей при освоении языка шаблонов. В этом им помогут следующие таблицы.

Терминология коммерческих систем обмена сообщениями

Шаблоны интеграции корпоративных приложений	Java Message Service (JMS)	Microsoft MSMQ	WebSphere MQ
Канал сообщений (***Message Channel***, с. 93)	Destination	MessageQueue	Queue
Канал "точка-точка" (***Point-to-Point Channel***, с. 131)	Queue	MessageQueue	Queue
Канал "публикация-подписка" (***Publish-Subscribe Channel***, с. 134)	Topic	—	—
Сообщение (***Message***, с. 98)	Message	Message	Message
Конечная точка сообщения (***Message Endpoint***, с. 124)	MessageProducer, MessageConsumer	—	—

Терминология коммерческих систем обмена сообщениями

Шаблоны интеграции корпоративных приложений	TIBCO	WebMethods	SeeBeyond	Vitria
Канал сообщений (***Message Channel***, с. 93)	Subject	Queue	Intelligent Queue	Channel
Канал "точка-точка" (***Point-to-Point Channel***, с. 131)	Distributed Queue	Deliver Action	Intelligent Queue	Channel
Канал "публикация-подписка" (***Publish-Subscribe Channel***, с. 134)	Subject	Publish-Subscribe Action	Intelligent Queue	Publish-Subscribe Channel
Сообщение (***Message***, с. 98)	Message	Document	Event	Event
Конечная точка сообщения (***Message Endpoint***, с. 124)	Publisher, Subscriber	Publisher, Subscriber	Publisher, Subscriber	Publisher, Subscriber

Форма шаблонов

Идея использования шаблонов для документирования приемов программирования была популяризирована благодаря таким книгам, как *Design Patterns*, *Pattern Oriented Software Architecture*, *Core J2EE Patterns* и *Patterns of Enterprise Application Architecture* (Фаулер М. *Архитектура корпоративных программных приложений*. — М. : Издательский дом "Вильямс", 2004). Впервые концепция шаблонов и языка шаблонов была представлена в книгах Кристофера Александера (Christopher Alexander) *A Pattern Language* и *A Timeless Way of Building*. Каждый шаблон представляет собой некоторое решение, которое должно быть принято на этапе проектирования, а также его обоснование. Язык шаблонов — это набор тесно связанных между собой шаблонов. Применение шаблонов проектирования является одним из наиболее эффективных способов документирования экспертных знаний.

Язык шаблонов позволяет решать неограниченное число задач в рамках конкретной проблемной области. Как правило, каждая решаемая задача предполагает уникальный выбор и способ использования шаблонов. Эта книга поможет вам найти верный ответ на любой вопрос, связанный с применением технологии обмена сообщениями.

Использование формы шаблонов не гарантирует успех книги. Недостаточно сказать: "Для решения этой задачи примените такой-то шаблон". Шаблон представляет ценность только в том случае, если он содержит обоснование сложности задачи, описание возможных способов ее решения и объяснение преимуществ предлагаемого подхода. Кроме того, шаблоны должны быть связаны друг с другом для наглядной иллюстрации взаимопроникновения проблем проектирования. В этом случае форма шаблонов не только подскажет способ решения конкретной задачи, но и поможет выработать подход к успешному преодолению новых, не предусмотренных авторами этой книги, проблем.

В некотором смысле шаблоны подобны директивам. Они не описывают проблему или ее решение — вместо этого шаблоны *учат* решать проблему. Каждый шаблон представляет собой вопрос, ответ на который необходимо дать разработчику, например "Стоит ли использовать *обмен сообщениями*?" или "Будет ли оправдана в данной ситуации отправка *сообщения с командой*?". Суть шаблонов и языка шаблонов заключается в способствовании принятию правильных решений в любой (даже не предусмотренной автором шаблона) ситуации.

Универсальной формы шаблона не существует. Избранный нами стиль очень близок к форме Александера, ставшей популярной благодаря книге Кента Бека (Kent Beck) *Smalltalk Best Practice Patterns*. Для улучшения восприятия материала мы использовали такие элементы стилевого оформления, как подчеркивание, курсив, а также наглядные иллюстрации.

Форма каждого шаблона, представленного в этой книге, состоит из следующих элементов.

- **Имя.** Имя шаблона отражает его назначение. Основным критерием, использованным при выборе имени шаблона, была простота употребления этого имени в предложениях, а значит, и в разговорах между проектировщиками.
- **Пиктограмма.** В добавок к имени большинство шаблонов имеет пиктограмму. Необходимость использования пиктограмм была обусловлена тем, что многие архитекторы привыкли работать с диаграммами, т.е. с визуальным представлением шаблонов. К тому же применение пиктограмм позволяет наглядно представить композицию нескольких шаблонов.
- **Контекст.** В этом разделе описывается ситуация, которая привела к возникновению задачи. Зачастую в контексте упоминаются другие шаблоны проектирования.
- **Задача.** Постановка задачи представлена в виде вопросительного предложения, ограниченного двумя горизонтальными линиями.
- **Оценка сложности.** В этом разделе описываются ограничения, затрудняющие решение задачи. Также здесь могут упоминаться альтернативные решения, применение которых по тем или иным причинам нецелесообразно.
- **Решение.** Описание действий, которые необходимо предпринять для решения задачи. Постановка задачи и ее решение являются ключевыми компонентами шаблона. Чтобы облегчить повторное использование книги, для выделения текста постановки и решения задачи используется одинаковое стилевое оформление.

- **Эскиз.** Одной из наиболее привлекательных особенностей формы Александера является наличие эскиза, иллюстрирующего решение задачи. В большинстве случаев для понимания сути шаблона достаточно взглянуть на его имя и эскиз решения. Последовав примеру, мы разместили эскиз решения непосредственно после его описания для каждого шаблона.
- **Результат.** В этой части рассматриваются особенности применения решения, а также трудности, которые могут при этом возникнуть.
- **Что дальше.** В этом разделе перечисляются шаблоны, на которые стоит обратить внимание после применения решения. Как правило, использование одного из шаблонов ведет к возникновению новых трудностей, для устранения которых следует обратиться к другим шаблонам. Высокая степень взаимопроникновения шаблонов — одна из основных особенностей, отличающих язык шаблонов от простого каталога шаблонов.
- **Врезки.** Врезки содержат информацию о технических подробностях или разновидностях шаблона. Специфическое стилевое форматирование позволяет пропустить врезку, если она не касается интересующей вас реализации шаблона.
- **Примеры.** Обычно шаблон содержит один или несколько примеров его применения на практике. Пример реализации шаблона может содержать как одно лишь упоминание об известном способе его использования, так и большой сегмент программного кода. Учитывая разнообразие существующих технологий обмена сообщениями, мы не рассчитываем на то, что читатель будет знаком с каждой из них. Именно поэтому мы используем форму шаблона, позволяющую безболезненно пропустить примеры, не потеряв при этом важной информации.

Самое большое преимущество использования шаблонов состоит в том, что, помимо решения конкретной задачи, шаблон позволяет разрабатывать решения для новых, не предусмотренных его авторами задач. Таким образом, представленные в книге шаблоны могут быть использованы при работе не только с существующими системами обмена сообщениями, но и с системами обмена сообщениями, которые будут созданы после ее выхода в свет.

Диаграммы, использованные в книге

Интеграционные решения состоят из множества различных компонентов — приложений, баз данных, конечных точек, каналов, сообщений, маршрутизаторов и т.д. Чтобы описать интеграционное решение, следует определить нотацию, содержащую средства для представления вышеперечисленных компонентов. К сожалению, на сегодняшний день не существует всеобъемлющей, широко известной нотации, с помощью которой можно было бы описать все аспекты интеграционных решений. Унифицированный язык моделирования (Unified Modeling Language — UML) отлично подходит для описания объектно-ориентированных систем с помощью диаграмм классов и взаимодействия, однако он не содержит семантику для описания решений, основанных на обмене сообщениями. Профиль UML для интеграции корпоративных приложений (UML Profile for EAI — UMLEAI) [42] расширяет семантику диаграмм взаимодействия за счет поддержки

описания обмена сообщениями между компонентами системы. Эта нотация может служить отличной наглядной спецификацией для генерации кода как часть архитектуры, управляемой моделью (model-driven architecture — MDA). Тем не менее мы приняли решение отказаться от профиля UMLEAI по двум причинам. Во-первых, с помощью UMLEAI нельзя описать все шаблоны, представленные в этой книге. Во-вторых, нашей целью является не точная визуальная спецификация, а "эскиз" или "набросок", способный максимально просто и лаконично передать суть шаблона. Именно поэтому мы и решили разработать собственную "нотацию", не требующую от читателя изучать длинную документацию по ее использованию. Пример употребления созданной нами нотации представлен на рис. 3.

На рис. 3 изображено сообщение, передаваемое по каналу *компоненту* в самом широком смысле этого слова. Так, это может быть интегрируемое приложение; посредник, преобразующий сообщения или маршрутизирующий их между приложениями; определенная часть приложения. Если на рисунке необходимо подчеркнуть использование канала, он изображается в виде трехмерной трубки. Однако гораздо чаще акцент переносится на компоненты, а канал изображается в виде линии с указывающей стрелкой. Два различных способа представления канала на рисунке полностью эквивалентны. Как показано на рис. 3, сообщению соответствует небольшое дерево с круглым корнем и квадратными вложенными элементами. Подобный способ представления сообщения обусловливается использованием многими системами обмена сообщениями древовидных структур данных, таких как документы XML. Элементы дерева могут быть заштрихованы или залиты определенным цветом для подчеркивания способа их использования в конкретном шаблоне. В частности, это позволяет создать наглядное визуальное представление для шаблонов преобразования, предназначенных для добавления, изменения порядка следования или удаления полей сообщения.

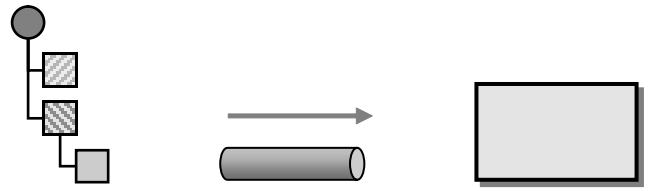

Рис. 3. Пример визуального представления компонентов интеграционного решения

Описывая структуру приложения, мы не используем стандартные диаграммы классов и циклограммы UML для представления иерархии классов и взаимодействия между объектами. Одним из наиболее авторитетных источников информации по нотации UML является [41].

Примеры и практикумы

Стараясь подчеркнуть широкую применимость шаблонов проектирования, представленных в этой книге, мы включили в нее множество примеров использования различных технологий интеграции. Недостаток подобного подхода состоит в том, что читатель может не быть знаком с каждой из упомянутых нами технологий. Именно поэтому мы постарались сделать все примеры исключительно факультативными — в них не содержится важной информации, касающейся шаблона. Там, где это было возможно, мы приводили несколько примеров реализации с использованием различных технологий интеграции.

Включая в текст программный код, мы прежде всего заботились о его *удобочитаемости*, а не о *возможности выполнения*. Фрагмент кода способен снять все вопросы, касающиеся шаблона. Именно поэтому многие разработчики и архитекторы предпочитают просмотреть несколько десятков строк кода, а не длинные параграфы текста. Чтобы облегчить восприятие кода, мы приводили только наиболее важные методы и классы решения, а также избегали большинства форм проверки ошибок. Практически все фрагменты кода не содержат комментариев, поскольку код обсуждается в прилегающих к нему абзацах текста.

Приведение полноценного примера использования шаблона интеграционного решения весьма затруднительно. Как правило, подобные решения состоят из множества гетерогенных компонентов, распределенных между различными системами. К тому же большинство интеграционных шаблонов тесно связаны с другими шаблонами. Чтобы продемонстрировать взаимозависимость нескольких шаблонов, мы включили в книгу несколько сложных примеров в виде так называемых “практикумов” (главы 6, 9 и 12). Рассмотренные нами примеры иллюстрируют многие проблемы проектирования сложных решений, основанных на обмене сообщениями.

Хотим подчеркнуть, что все примеры, приведенные в книге, носят иллюстративный характер и не могут применяться в качестве отправной точки при создании интеграционного решения, пригодного для применения в реальных условиях. В частности, практически во всех примерах отсутствует проверка ошибок, а также средства обеспечения надежности, безопасности и масштабируемости.

При наличии такой возможности мы старались использовать в примерах бесплатные программные платформы или же платформы, для которых имеется общедоступная пробная версия. Иногда мы применяли коммерческие платформы (такие, как TIBCO ActiveEnterprise и Microsoft BizTalk) с целью продемонстрировать различие между использованием платных программ и разработкой решения “с нуля”. В подобных случаях мы старались сделать так, чтобы читатель извлек пользу из примера, даже не имея под рукой необходимых программных средств. Большинство примеров основано на использовании таких инфраструктур для обмена сообщениями, как JMS и MSMQ. Это позволяет в полной мере сфокусироваться на существующих проблемах реализации интеграционных решений вместо того, чтобы пытаться избежать их с помощью более функционального связующего ПО.

Все Java-примеры, приведенные в этой книге, основаны на спецификации JMS 1.1, включенной в версию 1.4 платформы J2EE. Ко времени опубликования книги большинство серверов обмена сообщениями и серверов приложений будут поддерживать JMS 1.1.

Эталонную реализацию JMS можно загрузить с Web-сайта компании Sun Microsystems по адресу `http://java.sun.com/j2ee`.

Также в книге приводятся примеры, основанные на версии 1.1 платформы Microsoft .NET и написанные на языке программирования C#. Набор инструментальных средств разработки для платформы .NET Framework можно загрузить с Web-сайта корпорации Microsoft по адресу `http://msdn.microsoft.com/net`.

Как организована эта книга

Язык шаблонов, представленный в этой книге, представляет собой множество взаимосвязанных шаблонов проектирования. В то же время в языке шаблонов можно выделить несколько ключевых элементов (так называемых *корневых шаблонов*), формирующих его логическую структуру.

При группировании шаблонов по главам этой книги учитывался их уровень абстракции, а также область применения (рис. 4).

Основным шаблоном проектирования книги является шаблон *обмен сообщениями* (*Messaging*, с. 87). К корневым шаблонам относятся шаблоны *канал сообщений* (*Message Channel*, с. 93), *сообщение* (*Message*, с. 98), *каналы и фильтры* (*Pipes and Filters*, с. 102), *маршрутизатор сообщений* (*Message Router*, с. 109), *транслятор сообщений* (*Message Translator*, с. 115) и *конечная точка сообщения* (*Message Endpoint*, с. 124). Каждый из корневых шаблонов рассматривается в отдельной главе. Исключение составляет шаблон *каналы и фильтры*; рассматриваемая в нем концепция не имеет прямого отношения к обмену сообщениями, а формирует основу для шаблонов маршрутизации и преобразования.

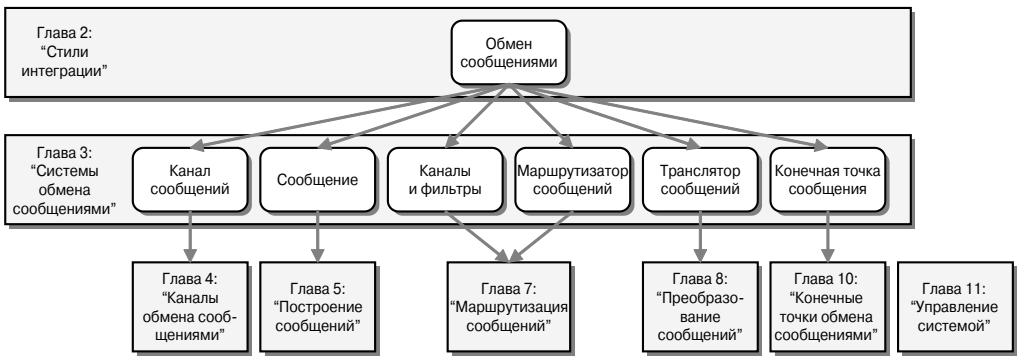

Рис. 4. Корневые шаблоны проектирования

Описанная выше иерархия шаблонов проектирования рассматривается в следующих главах.

- Глава 2. В этой главе рассматриваются различные подходы к интеграции приложений, включая *обмен сообщениями*.
- Глава 3. В этой главе рассматриваются шесть фундаментальных шаблонов проектирования, представленных в книге.

- Глава 4. Приложения взаимодействуют между собой посредством каналов. Каналы определяют логические маршруты, по которым могут передаваться сообщения. Эта глава посвящена выбору каналов обмена сообщениями для конкретного приложения.
- Глава 5. В этой главе рассматриваются различные форматы сообщений и их свойства.
- Глава 7. Решения, основанные на обмене сообщениями, предполагают разделение отправителя и получателя информации. Отправитель пересылает сообщение маршрутизатору, который обеспечивает его доставку получателю. В этой главе рассматриваются различные способы создания маршрутизаторов сообщений.
- Глава 8. Зачастую независимо созданные приложения используют различные форматы сообщений, кодировку символов и т.п. В этой главе рассматривается создание промежуточных компонентов, преобразующих формат отправителя в формат получателя сообщения.
- Глава 10. Многие приложения изначально не приспособлены для участия в системах обмена сообщениями. В этой главе рассматривается уровень, обеспечивающий отправку и получение сообщений, который тем самым превращает приложение в конечную точку системы обмена сообщениями.
- Глава 11. Эта глава посвящена тестированию и мониторингу системы обмена сообщениями.

Представленные выше главы содержат всю необходимую информацию, касающуюся интеграции приложений с использованием обмена сообщениями.

С чего начать

Приступая к работе с книгой, охватывающей обширную область знаний, важно определить наиболее эффективный способ ее прочтения. Несмотря на то что метод чтения “от корки до корки” гарантирует изучение всего материала, представленного в настоящей книге, это далеко не самый быстрый способ добраться до нужной темы. Начинать чтение книги с произвольного шаблона — также не самая удачная идея: в этом случае вы рискуете упустить важную информацию, необходимую для понимания шаблона.

К счастью, представленный в книге язык шаблонов построен вокруг упоминавшихся выше корневых шаблонов. Все вместе корневые шаблоны позволяют получить наглядное представление о языке шаблонов, а по отдельности представляют собой прекрасные отправные точки для более глубокого изучения технологий обмена сообщениями. Описание всех корневых шаблонов содержится в главе 3.

В главе 2 рассматриваются основные подходы к интеграции приложений, включая *обмен сообщениями* (*Messaging*, с. 87). Прочитайте эту главу, если вы впервые столкнулись с проблемой интеграции приложений и хотите узнать о преимуществах и недостатках существующих интеграционных технологий. В противном случае вы можете пропустить главу 2.

Глава 3 содержит описание всех корневых шаблонов, за исключением шаблона *обмен сообщениями*, который рассматривается в главе 2. Прочитайте (или хотя бы просмотрите) главу 3, чтобы получить общее представление об используемом в книге языке шаблонов. Для более глубокого изучения той или иной темы ознакомьтесь с соответствующим кор-

невым шаблоном, после чего переходите к рассмотрению шаблонов, перечисленных в описании этого корневого шаблона.

Начиная с главы 4, различным участникам процесса интеграции приложений предлагается обратить внимание на следующий материал.

- *Системным администраторам* рекомендуется обратить внимание на тему выбора каналов обмена сообщениями (глава 4) и обслуживания системы обмена сообщениями (глава 11).
- *Разработчики приложений* должны ознакомиться с вопросом интеграции приложения с системой обмена сообщениями (глава 10) и выбора формата отправляемых сообщений (глава 5).
- *Системные интеграторы* извлекут максимум пользы из чтения главы 7, посвященной доставке сообщений в конечные точки, и главы 8, описывающей преобразование формата отправителя в формат получателя сообщения.

Если вам нужно найти шаблон, подходящий для применения в конкретной ситуации, обращайте внимание только на постановку задачи и ее решение. Этой информации будет вполне достаточно, чтобы понять, интересует вас данный шаблон или нет.

Обратите внимание, что порядок следования шаблонов в книге не определяет рекомендуемый порядок их изучения. Столкнувшись с той или иной задачей интеграции, выберите соответствующий ей корневой шаблон. Из его контекста вы узнаете, какие шаблоны следует применить перед использованием данного корневого шаблона (их описания могут располагаться как до, так и после описания корневого шаблона), а из раздела, предшествующего примерам, — какие шаблоны следует применить после использования данного корневого шаблона (опять-таки, описания этих шаблонов необязательно будут следовать после описания корневого шаблона). Другими словами, эта книга представляет собой "паутину" взаимосвязанных шаблонов, а не их последовательное описание.

Поддержка

Дополнительные материалы к книге, а также информация, касающаяся интеграции корпоративных приложений, доступна в Интернете на Web-сайте `www.enterpriseintegrationpatterns.com`. Комментарии, пожелания и отзывы присылайте по адресу `authors@enterpriseintegrationpatterns.com`.

Резюме

Назначение этого введения — ознакомить читателей со следующими фактами и фундаментальными концепциями:

- что такое обмен сообщениями;
- что такое система обмена сообщениями;
- преимущества технологии обмена сообщениями;
- отличия асинхронного и синхронного программирования;

- отличия интеграции приложений от создания распределенного приложения;
- существующие категории коммерческих программных продуктов, поддерживающих обмен сообщениями.

Кроме того, во введении рассматривалась организация книги и способ подачи материала. В частности, особое внимание было уделено:

- роли шаблонов в структурировании материала книги;
- нотации, использующейся для описания интеграционных решений;
- назначению и области применения приведенных в книге примеров;
- организации материала;
- определению наиболее эффективного способа изучения материала.

Ознакомившись с базовыми понятиями и способом организации материала, приступим к изучению интеграции корпоративных приложений с использованием обмена сообщениями.