Arnon Rotem-Gal-Oz

WZORCE SOA

Najlepsze podejście do wytwarzania oprogramowania!

MANNING

Helion

Tytuł oryginału: SOA Patterns

Tłumaczenie: Lech Lachowski
Projekt okładki: Anna Mitka

ISBN: 978-83-246-7050-5

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.
Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/wzosoa.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/wzosoa
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

5 Wzorce dotyczące wymiany komunikatów

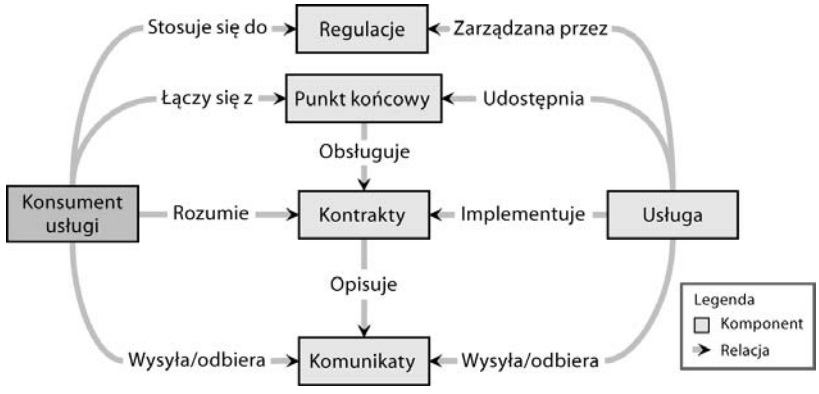
W tym rozdziale

- Interakcje pomiędzy usługami i konsumentami
- Komunikaty skorelowane
- Czas życia zdarzeń

W rozdziałach 2. i 3. przyjrzeliśmy się wzorcom takim jak Komponent Brzegowy i Instancja Usługi, które pomagają budować usługi i ich interfejsy. Rozdział 4. został poświęcony sposobom ochrony i monitorowania usług. Rozdział 5. jest pierwszym z trzech kolejnych rozdziałów omawiających różne aspekty interakcji usług. W końcu to zapewnienie interakcji usług i umożliwienie procesów biznesowych było pierwszym powodem do zastosowania SOA.

Jak pokazano na rysunku 5.1, rozdział ten skupia się na interakcjach usług z ich „klientami" — **konsumentami usług**. Konsument usług to dowolny komponent lub fragment kodu, który współdziała z usługą. Wzorce omówione w tym rozdziale dotyczą podstaw — wymiany komunikatów. Rozdział 6. został poświęcony konsumentom usług, a rozdział 7. koncentruje się na wzorcach związanych z kompozycją i integracją usług.

Definicja SOA z rozdziału 1. mówi, że „każda usługa ujawnia określone procesy i zachowania poprzez kontrakty, które składają się z komunikatów w wykrywalnych adresach". To znacznie upraszcza interakcje usług — po prostu wysyłasz komunikat i otrzymujesz komunikat zwrotny, prawda? Dlaczego więc musimy poświęcić interakcjom usług cały rozdział, a nawet dwa?



Rysunek 5.1. Rozdział ten koncentruje się na łączeniu usług z interfejsami użytkowników. To pierwszy z rozdziałów tej książki poświęcony konsumentom usług

To prawda, że komunikaty są podstawowym budulcem interakcji usług, ale istnieje wiele sposobów interakcji wykorzystujących ten budulec. Ludzie podobnie używają zdań jako budulca dla komunikacji i interakcji. Kiedy kontaktujesz się z przedstawicielem handlowym, możliwych jest kilka interakcji:

- Możesz zadać określone pytanie i uzyskać odpowiedź (wzorzec Żądanie/Odpowiedź z podrozdziału 5.1).
- Możesz zostawić wiadomość z zapytaniem i swoim numerem telefonu, a przedstawiciel handlowy oddzwoni do Ciebie później (wzorzec Żądanie/Reakcja z podrozdziału 5.2).
- Przedstawiciel handlowy może zadzwonić do Ciebie i poinformować Cię o nowych produktach (wzorzec Odwrócenie Komunikacji z podrozdziału 5.3).
- Możesz prowadzić obszerną korespondencję z przedstawicielem handlowym, wymieniając wiadomości e-mail, zanim Twój problem zostanie rozwiązany (wzorzec Saga z podrozdziału 5.4).

To, co dotyczy prawdziwego życia, sprawdza się również w przypadku usług.

W przeciwieństwie do większości innych wzorców z tej książki, te wzorce podstawowych interakcji istniały, zanim nawet wymyślono SOA — zadaniem tego rozdziału jest przyjrzenie się tym wzorcom interakcji z perspektywy SOA i atrybutów jakościowych tej architektury. Zobaczymy, co sprawia, że wzorzec interakcji, taki jak komunikacja asynchroniczna, działa zgodnie z zasadami SOA i zachowuje korzyści SOA.

W tym rozdziale zajmiemy się następującymi wzorcami:

- **Żądanie/Odpowiedź** (ang. *Request/Reply*) — Umożliwia konsumentowi usług prostą interakcję z usługą.
- **Żądanie/Reakcja** (ang. *Request/Reaction*) — Tymczasowo oddziela żądanie od konsumenta usług oraz odpowiedź od usługi.

- **Odwrócenie Komunikacji** (ang. *Inversion of Communications*) — Obsługuje zdarzenia biznesowe w SOA.
- **Saga** (ang. *Saga*) — Umożliwia osiągnięcie konsensusu rozproszonego pomiędzy usługami bez stosowania transakcji.

Zacznijmy od najbardziej podstawowej formy komunikacji — komunikacji synchronicznej. Wzorzec nazywa się Żądanie/Odpowiedź.

5.1. Wzorzec Żądanie/Odpowiedź

Żądanie/Odpowiedź jest prawdopodobnie najstarszym i najlepiej opisanym wzorcem w naukach komputerowych. Gregor Hohpe i Bobby Woolf oferują dobry opis wzorca Żądanie/Odpowiedź w swojej książce *Enterprise Integration Patterns* (Addison-Wesley Professional, 2003), gdzie określają ten wzorzec jako odpowiadający na pytanie: „Kiedy aplikacja wysyła komunikat, w jaki sposób może otrzymać odpowiedź od odbiorcy?".

Koncepcja wzorca Żądanie/Odpowiedź w SOA nie różni się znacznie. Powodem do omówienia danego wzorca w tej książce jest jednak to, że nadal istnieje kilka kwestii, na które warto zwrócić uwagę, korzystając z niego w SOA. Wspomnę o nich w kontekście omawiania rozwiązania. Najpierw przyjrzyjmy się problemowi.

PROBLEM

Podczas opracowywania jednowarstwowego oprogramowania, które działa wewnątrz jednego procesu i pojedynczej przestrzeni pamięci, stosunkowo łatwo jest uzyskać interakcję komponentów. Kiedy komponent żądający potrzebuje czegoś od innego komponentu (podmiotu odpowiadającego), może łatwo uzyskać referencję do tego komponentu, np. poprzez jego instancjonowanie. Podmiot żądający może wtedy wywołać metodę w komponencie odpowiadającym i otrzymać odpowiedź jako referencję lub adres pamięci, pod którym ta odpowiedź się znajduje.

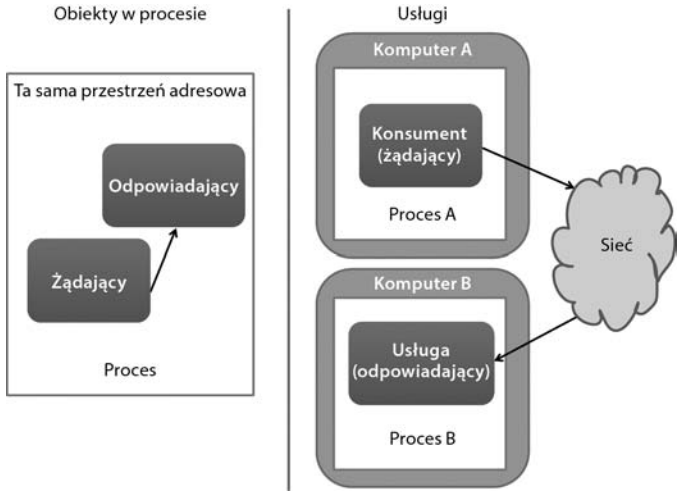
W SOA, które jest stylem architektonicznym systemów rozproszonych, drugi komponent zazwyczaj znajduje się w innej przestrzeni pamięci i najprawdopodobniej na innej maszynie — patrz rysunek 5.2.

> **UWAGA** Zdalne wywołania zostały technologicznie rozwiązane przed powstaniem SOA, ale były to rozwiązania przeznaczone dla innych stylów architektonicznych. Większość z tych technologii może być również wykorzystana w SOA — różnicę stanowi sposób ich wykorzystania. Wrócimy do tego w dalszej części rozdziału.

Pierwszą rzeczą, jaką musimy zrobić, jest znalezienie sposobu na interakcję usług z ich konsumentami.

? **Jak możesz umożliwić konsumentom usług prostą interakcję z usługą?**

W tym rozdziale zostało uszczegółowionych kilka alternatywnych sposobów interakcji z usługami: asynchroniczne Żądanie/Odpowiedź (wzorzec Żądanie/Reakcja),



Rysunek 5.2. Obiekty zinstancjonowane w obrębie jednego procesu w porównaniu z usługami. W przypadku obiektu lokalnego wysłanie żądania z jednego komponentu do drugiego jest proste — otrzymujesz referencję do tego drugiego komponentu i wykonujesz żądanie, wywołując go. W SOA żądający i konsument nie znajdują się w jednej przestrzeni adresowej. Prawdopodobnie nie znajdują się również na jednym komputerze, a może są nawet w innej sieci LAN. Wykonanie żądania w tych warunkach jest znacznie bardziej skomplikowane

długotrwałe interakcje (wzorzec Saga) lub zdarzenia (wzorzec Odwrócenie Komunikacji). Są one bardziej wydajne niż wzorzec Żądanie/Odpowiedź, ale ma to również swoją cenę — są one bardziej złożone od wzorca Żądanie/Odpowiedź zarówno w zakresie implementacji, jak i wsparcia.

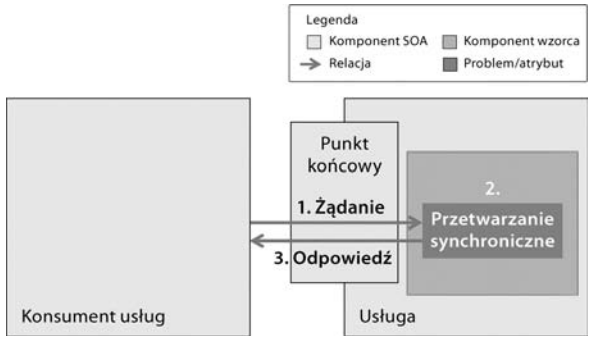
ROZWIĄZANIE

Wyrafinowanie bywa uzasadnione, ale czasem potrzebna jest prosta synchroniczna interakcja pomiędzy dwoma zdalnymi komponentami.

✓ **Wyślij żądanie od konsumenta, obsłuż żądanie synchronicznie i wyślij z usługi komunikat z odpowiedzią. Zarówno żądanie, jak i odpowiedź należą do usługi odbierającej.**

Przedstawiony na rysunku 5.3 wzorzec Żądanie/Odpowiedź jest najbardziej podstawowym wzorcem interakcji, nie wymaga więc żadnych specjalnych komponentów. Potrzebujesz jedynie schematu logicznego, który akceptuje żądania, przetwarza je synchronicznie i zwraca odpowiedź lub wynik. Jedną rzeczą, na którą należy zwrócić uwagę, jest to, że zarówno komunikat żądania, jak i komunikat z odpowiedzią należą do kontraktu usługi, a nie konsumenta usług (co jest typowym błędem u nowicjuszy w dziedzinie SOA).

Wzorzec Żądanie/Odpowiedź obejmuje jedynie wymianę komunikatów. Kompletna interakcja wymaga również infrastruktury komunikacyjnej. Mógłbyś w tym



Rysunek 5.3. Wzorzec Żądanie/Odpowiedź definiuje komunikaty żądań i odpowiedzi w kontrakcie usługi. Kiedy usługa otrzymuje żądanie w odpowiednim formacie, przetwarza je synchronicznie i zwraca konsumentowi usług komunikat z odpowiedzią

celu wykorzystać wzorzec Magistrala Usługi (omówiony w rozdziale 7.), który obsługuje udostępnianie usług w osiągalnych (lub nawet wykrywalnych) punktach końcowych, a także routuje odpowiedzi.

Role żądania i odpowiedzi są raczej oczywiste. Żądanie przechowuje zamiar lub zadanie, które usługa ma wykonać, a także dane wejściowe niezbędne do jego wykonania. Odpowiedź zawiera wynik wykonania zadania.

Głównym problemem ze stylem interakcji Żądanie/Odpowiedź jest to, że podejrzanie przypomina on zdalne wywołania procedur (RPC) — DCOM/CORBA, czyli rozproszone modele obiektowe. Powinieneś być ostrożny przy modelowaniu kontraktów usług z nastawieniem na RPC — może mieć to niekorzystny wpływ na Twoje systemy SOA, poczynając od niskiej wydajności, aż po całkowite zaprzeczenie tej architektury. Zamiast stosowania podejścia RPC możesz spróbować modelować swoje kontrakty metodą dokumentowo-centryczną. Czym, u licha, jest „podejście dokumentowo-centryczne"? Dobre pytanie.

W skrócie podejście **dokumentowo-centryczne** (ang. *document-centric*) oznacza, że komunikat zawiera wystarczającą ilość informacji, aby reprezentować kompletną jednostkę pracy, oraz nie instruuje usługi, w jaki sposób ma ona obsługiwać komunikat. Dla kontrastu wywołania RPC mają tendencję do bycia zorientowanymi na polecenia i nastawionymi na wysyłanie tylko parametrów niezbędnych do wykonania działania. Mają one pewne stanowe oczekiwania ze strony usługi oraz domniemane oczekiwania odnośnie tego, co ma stać się po stronie konsumenta. Dokumentowo-centryczne komunikaty nie robią takich założeń. Posiadanie kompletnej jednostki pracy oznacza, że usługa posiada wystarczającą ilość informacji lub kontekstu w komunikacie, aby zrozumieć wymagany stan. Oznacza to również, że komunikaty dokumentowo-centryczne są zazwyczaj bardziej gruboziarniste niż ich odpowiedniki RPC.

> **UWAGA** Jest jeszcze trzeci typ komunikatów, zwany **komunikatami zdarzeń** (ang. *event messages*). Zajmiemy się nim w kontekście wzorca Odwrócenie Komunikacji w podrozdziale 5.3.

W tabeli 5.1 przedstawiono trzy sposoby na to, aby dokumentowo-centryczne komunikaty mogły zawierać więcej kontekstu.

Tabela 5.1. Opcje dostarczenia kontekstu w komunikacie dokumentowo-centrycznym

Kontekst	Wyjaśnienie
Historia	Komunikaty mogą zawierać interakcje do określonego momentu, podobnie jak okruszki chleba w bajce o Jasiu i Małgosi. Jeśli w scenariuszu zamawiania pierwszym krokiem było pobranie danych konsumenta, a bieżącym krokiem jest złożenie zamówienia (każda czynność wykonywana przez inną usługę), to komunikat przesyłany do usługi zamawiania zawierałby informacje o konsumencie
Przyszłość	Komunikat może zawierać opcje, z których konsument może skorzystać w celu wypełnienia interakcji. Jeśli w scenariuszu zamawiania poprzednim krokiem było zarezerwowanie zamówienia (patrz wzorzec Rezerwacja w rozdziale 6.), komunikat zwrotny mógłby zawierać informacje wymagane do potwierdzenia rezerwacji
Kompletna przyszłość	Innym sposobem zapewnienia kontekstu jest, aby format komunikatu zawierał kompletne dane szczegółowe niezbędne do przeprowadzenia interakcji. W przykładzie zamawiania oznaczałoby to, że komunikat posiadałby szkielet do obsługi szczegółów zamówienia i kwestii z nim związanych, a zaangażowane strony wypełniałyby puste pola wraz z postępowaniem interakcji

Należy zwrócić uwagę na dwie kwestie. Komunikaty mogą łączyć kilka typów kontekstu, a ten sam dokument, w celu umożliwienia zakończenia procesu biznesowego, może być wymieniany wielokrotnie pomiędzy usługą i jej konsumentem, np. z uwagi na dodawanie na tej drodze różnych szczegółów.

MAPOWANIE TECHNOLOGII

Mapowanie technologii dla wzorca Żądanie/Odpowiedź jest raczej trywialne. Każda z przychodzących mi do głowy technologii umożliwia implementację tego wzorca w tej czy innej formie.

Większość technologii nadzwyczaj łatwo pozwala udostępniać zdalnie obiekty, co zachęca do zastosowania interakcji w stylu RPC. Interakcje dokumentowo-centryczne są znacznie trudniejsze do wprowadzenia. Listing 5.1 przedstawia fragment kodu z kreatora nowego projektu dla biblioteki usług WFC w Visual Studio 2010 Microsoftu. Przykładowy kod pokazuje, jak wziąć prostą klasę i udostępnić jej metody jako usługi sieciowe.

Listing 5.1. Kod z kreatora nowego projektu służący tworzeniu usługi WFC

```
namespace WCFServiceLibrary1
{
    [ServiceContract()]
    public interface IService1
    {
        [OperationContract]          <-- Udostępnia metodę jako usługę sieciową
        string MyOperation1(string myValue);
        [OperationContract]
        string MyOperation2(DataContract1 dataContrac tValue);
    }

    public class service1 : IService1
    {
        public string MyOperation1(string myValue)
        {
```

```
        return "Hello: " + myValue;
    }
    public string MyOperation2(DataContract1 dataContractValue)
    {
        return "Hello: " + dataContractValue.FirstName;
    }
}

[DataContract]
public class DataContract1
{
    string firstName;
    string lastName;

    [DataMember]
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    [DataMember]
    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
}
```

Akceptuje dokument (kontrakt danych) jako parametr

Obsługuje dokument na sposób RPC (nie zwraca dokumentu)

Definiuje podstawowy dokument (brakujące łącza do powiązanych danych)

Na pierwszy rzut oka ten kod może wyglądać jak dobry przykład wzorca Żądanie/Odpowiedź (może z wyjątkiem nazewnictwa). Konsument usługi może wysłać komunikat `MyOperation1` z załączonym ciągiem znaków i otrzymać jako odpowiedź „Hello:" dołączone do tego ciągu. Jednak `MyOperation1` to klasyczna implementacja interakcji RPC.

Sytuacja jest nieco lepsza w przypadku drugiej metody (`MyOperation2`). Tutaj prosty dokument jest przekazywany do metody. Jednak ten przykładowy kod obsługuje dany dokument również na sposób RPC i nie zwraca dokumentu w ramach odpowiedzi.

Takie podejście nie jest unikatowe dla platformy .NET — kolejnym przykładem może być styl REST. O ile zasady REST promują podejście dokumentowo-centryczne, to podstawowymi poleceniami HTTP są `PUT`, `GET`, `POST` i `DELETE`, które ponownie kierują nowicjuszy na tory interfejsów CRUD.

Dokumentowo zorientowane podejście skutkuje bogatszymi komunikatami, które zawierają pewien kontekst, jeśli nawet nie cały. Przyjrzyj się fragmentowi kodu XML z listingu 5.2.

Listing 5.2. Przykładowa odpowiedź dokumentowo-centryczna

```
<feed xmlns='http://www.w3.org/2005/Atom'
    xmlns:gd='http://schemas.google.com/g/2005'>
<id>http://www.google.com/calendar/feeds/johndoe@gmail.com/
➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full</id>
  <updated>2007-06-29T19:22:12.000Z</updated>
```

```
  <title type='text'>John Doe</title>
  <link rel='http://schemas.google.com/g/2005#feed' type='application/atom+xml'
    href='http://www.google.com/calendar/feeds/johndoe@gmail.com/
    ➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full'></link>
  <link rel='self' type='application/atom+xml'
    href='http://www.google.com/calendar/feeds/johndoe@gmail.com/
    ➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full'></link>
  <author>
    <name>John Doe</name>
    <email>johndoe@gmail.com</email>
  </author>
  <generator version='1.0' uri='http://www.google.com/calendar/'>
  CL2
  </generator>
  <gd:where valueString='Neverneverland'></gd:where>
  <entry>
    <id>http://www.google.com/calendar/feeds/johndoe@gmail.com/
    ➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full/
    ➥aaBxcnNqbW9tcTJnaTT5cnMybmEwaW04bXMgbWFyY2guam9AZ21haWwuY29t</id>
    <published>2007-06-30T22:00:00.000Z</published>
    <updated>2007-06-28T015:33:31.000Z</updated>
    <category scheme='http://schemas.google.com/g/2005#kind'
      term='http://schemas.google.com/g/2007#event'></category>
    <title type='text'>Writing SOA Patterns</title>
    <content type='text'>shhh…</content>
    <link rel='alternate' type='text/html'
      href='http://www.google.com/calendar/
      ➥event?eid=aaBxcnNqbW9tcTJnaTT5cnMybmEwaW04bXMgbWFyY2guam9AZ21haWwuY29t'
      title='alternate'></link>
    <link rel='self' type='application/atom+xml'
      href='http://www.google.com/calendar/feeds/johndoe@gmail.com/
      ➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full/
      ➥aaBxcnNqbW9tcTJnaTT5cnMybmEwaW04bXMgbWFyY2guam9AZ21haWwuY29t'></link>
    <author>
      <name>John Doe</name>
      <email>johndoe@gmail.com</email>
    </author>
    <gd:transparency
      value='http://schemas.google.com/g/2005#event.opaque'>
    </gd:transparency>
    <gd:eventStatus
      value='http://schemas.google.com/g/2005#event.confirmed'>
    </gd:eventStatus>
    <gd:comments>
      <gd:feedLink
        href='http://www.google.com/calendar/feeds/johndoe@gmail.com/
        ➥private-0c1e3facdd1a4252aad07effeb7d68cc9/full/
        ➥aaBxcnNqbW9tcTJnaTT5cnMybmEwaW04bXMgbWFyY2guam9AZ21haWwuY29t/comments/'>
      </gd:feedLink>
    </gd:comments>
    <gd:when startTime='2006-08-14T20:30:00.000Z'
       endTime='2012-03-28T22:30:00.000Z'></gd:when>
    <gd:where></gd:where>
  </entry>
</feed>
```

Ten listing pokazuje wynik żądania pełnego kalendarza z usługi Google Kalendarz. Poza szczegółami dotyczącymi kalendarza (tytuł, data aktualizacji, nazwa właściciela itd.) otrzymujesz wszystkie zestawienia ze wszystkimi szczegółami, a także wskaźnik do pobrania każdego wpisu kalendarza bezpośrednio. Wynik wykorzystuje protokół Google GData, który z kolei jest oparty na protokole APP (ang. *Atom Publishing Protocol*). Zwróć uwagę, że kontrakt dla akceptowania tego dokumentu XML jest również prostszy niż ten z listingu 5.1, ponieważ potrzeba jedynie obsłużyć pojedynczy parametr XML. Konsumenci nie są przywiązani do określonych informacji, które mogą zmieniać się z czasem.

Podsumowując, wzorzec Żądanie/Odpowiedź jest obsługiwany przez wszystkie technologie umożliwiające zdalną komunikację. Wybór pomiędzy RPC a podejściem dokumentowo-centrycznym jest decyzją projektową, która nie jest determinowana przez technologię. Musi ona zostać podjęta przez programistów lub architektów danego rozwiązania.

ATRYBUTY JAKOŚCIOWE

Wzorzec Żądanie/Odpowiedź jest prostym wzorcem łączącym konsumenta usług z usługą, z którą chce on wejść w interakcję. Będąc wzorcem podstawowym, nie rozwiązuje większości kwestii związanych z atrybutami jakościowymi, z wyjątkiem zapewnienia wymaganej funkcjonalności (umożliwienia interakcji pomiędzy konsumentem a usługą).

Jednym z atrybutów jakościowych, który może być istotny, jest prostota. Ponieważ Żądanie/Odpowiedź jest prostym wzorcem, jest łatwy w implementacji i zapewnieniu wsparcia, dzięki czemu pomaga zmniejszyć stopień złożoności rozwiązania.

Tabela 5.2 zawiera przykładowe scenariusze, w których można rozważyć zastosowanie wzorca Żądanie/Odpowiedź.

Tabela 5.2. Atrybuty jakościowe wzorca Żądanie/Odpowiedź i przykładowe scenariusze

Atrybut jakościowy	Konkretny atrybut	Przykładowy scenariusz
Czas opracowywania	Łatwy rozwój	Podczas fazy rozwoju udostępnianie nowych funkcjonalności (przygotowanych uprzednio) w usłudze nie powinno zająć więcej niż pół dnia, wliczając implementację i testy
Testowalność	Pokrycie	Podczas fazy rozwoju każda funkcjonalność usługi powinna mieć 100-procentowe pokrycie testowe

Jak wspomniano wcześniej, Żądanie/Odpowiedź jest podstawowym synchronicznym wzorcem komunikacyjnym. Następny wzorzec interakcji koncentruje się na implementacji komunikacji asynchronicznej w ramach ograniczeń i zasad SOA.

5.2. Wzorzec Żądanie/Reakcja

Komunikacja synchroniczna opisana w kontekście wzorca Żądanie/Odpowiedź (w poprzednim podrozdziale) jest bardzo istotna, ale nie jest wystarczająca. Synchroniczna natura wzorca Żądanie/Odpowiedź oznacza, że konsument usług musi oczekiwać, aż usługa zakończy przetwarzanie jego żądania, zanim będzie mógł

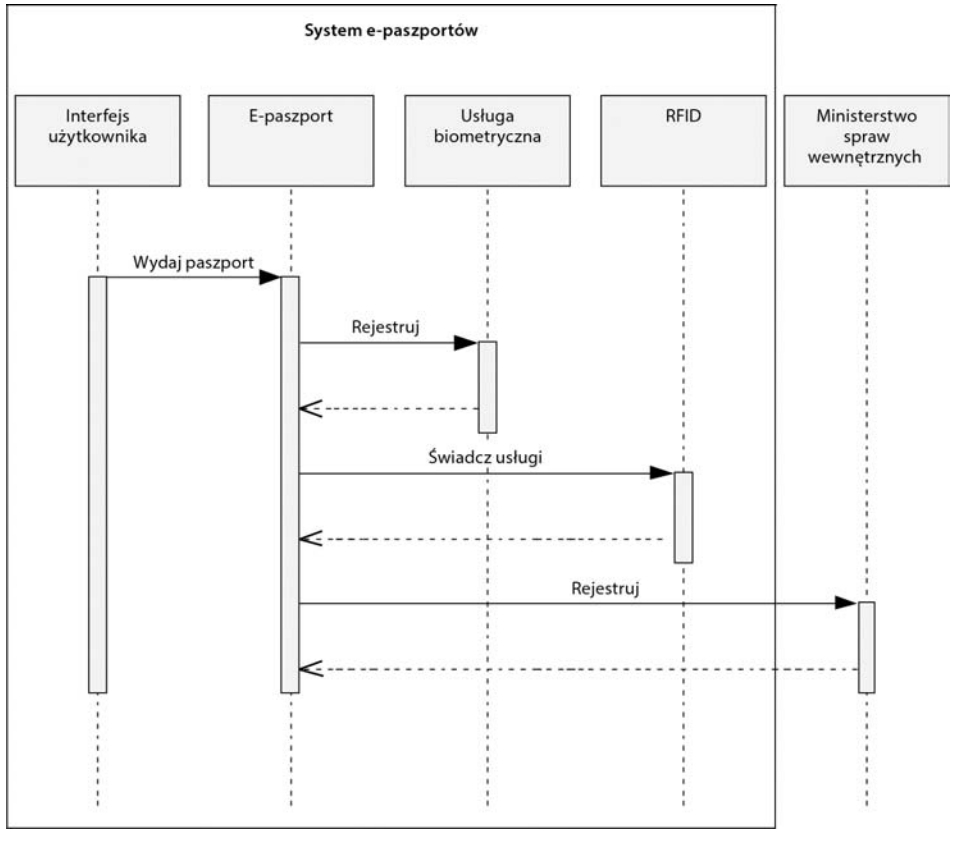
kontynuować swoje działanie. Są sytuacje, w których konsument usług nie chce czekać lub nie może sobie na to pozwolić, jednak nadal jest zainteresowany otrzymaniem odpowiedzi, kiedy tylko będzie ona dostępna.

Masło maślane? Przyjrzyjmy się konkretnemu przykładowi, abym mógł to lepiej wyjaśnić.

PROBLEM

We współczesnych systemach kontroli granic, kiedy podróżnik podchodzi do urzędnika imigracyjnego, urzędnik sprawdza w systemie szczegółowe informacje na temat danej osoby (wpisuje numer seryjny paszportu itd.), a następnie przegląda paszport i porównuje zdjęcie. W ciągu kilku ostatnich lat państwa na całym świecie zaczęły przechodzić na systemy e-paszportów. E-paszporty zawierają kilka elementów, w tym chip RFID, odczytywalny maszynowo kod oraz kilka próbek biometrycznych (zazwyczaj zdjęcie twarzy oraz odciski palców).

Na rysunku 5.4 przedstawiono wysokopoziomowy przegląd schematu przepływu dla wydawania e-paszportów.



Rysunek 5.4. Proces rejestracji. Kiedy interfejs użytkownika (UI) wysyła do usługi e-paszportu zapytanie o wydanie paszportu, usługa, aby spełnić to żądanie, musi współpracować z kilkoma innymi usługami

Jak widać, jednym z etapów schematu przepływu jest rejestrowanie osoby w bazie danych biometrii (która jest elementem usługi biometrycznej). Chociaż nie wynika to z samego schematu interakcji, zadanie rejestracji może zająć sporo czasu, ponieważ usługa biometryczna wewnętrznie sprawdza, czy istnieją duplikaty. Jest to niezbędne w celu zapewnienia integralności danych i uniknięcia błędów oraz celowych zmian tożsamości. Ten etap polega na porównywaniu każdej próbki (np. każdej twarzy) z każdą pozostałą próbką znajdującą się już w bazie danych, co może stanowić setki milionów rekordów (populacja całego kraju).

Wykonywanie tego typu żądań przy zastosowaniu wzorca interakcji Żądanie/Odpowiedź jest problematyczne, ponieważ czas oczekiwania pomiędzy żądaniem a odpowiedzią jest zbyt długi. Może być nawet jeszcze gorzej, jeśli zdecydujesz się na podwójne sprawdzanie w trakcie zautomatyzowanych nocnych procesów konserwacyjnych.

Taka sytuacja nie jest unikatowa dla systemów e-paszportów. Podobne sytuacje zdarzają się również w innych systemach. Przykładowo, przy zakupie udziałów w funduszu powierniczym transakcja nie jest przeprowadzana natychmiastowo, ale prawdopodobnie będziesz chciał wiedzieć, kiedy zostanie zakończona. Innym przykładem jest skierowanie żądania do systemu planowania podróży, żeby znalazł Ci najlepszą ofertę na kolejne wakacje. Oto problem:

? **Jak możesz tymczasowo oddzielić żądanie od konsumenta usług oraz odpowiedź od usługi?**

Jedną z opcji jest rozwiązanie kwestii tymczasowych powiązań po stronie klienta. W tym celu uruchamiany jest nowy wątek przed wysłaniem żądania do usługi. Następnie wątek oczekuje na odpowiedź, podczas gdy reszta interfejsu użytkownika pozostaje w stanie reagowania. Platforma .NET posiada komponent zwany BackgroundWorker, który dokonuje tej separacji i pozwala interfejsowi użytkownika na dysponowanie długotrwałych zadań bez blokowania swojego wątku.

To rozwiązanie ma swoje wady. Po pierwsze, „oczekiwanie" nie jest odporne — jeśli konsument usług będzie miał awarię, odpowiedź zostanie utracona i nie będzie dostępna, gdy konsument odzyska sprawność. Ponadto wątek wykorzystuje zasoby konsumenta — co się stanie, jeśli przetwarzanie żądania będzie trwało kilka godzin lub dni? Dochodzi jeszcze kwestia odpowiedzialności. To usługa ma do wykonania zadanie, które jest czasochłonne — odpowiedzialnością usługi powinno być rozwiązanie tego problemu i nieprzerzucanie go na konsumentów.

Innym podejściem rozwiązującym kwestię tymczasowego oddzielenia jest obejście tego problemu poprzez podzielenie interakcji. Przykładowo, kiedy zamawiasz towar online, nie siedzisz, czekając, aż system wyśle Ci ten towar. W zamian system powiadamia Cię, że towar został zamówiony. Rejestracja zamówienia zajmuje znacznie mniej czasu niż jego realizacja.

Minusem jest to, że nie będziesz wiedział, czy towar został wysłany, jeśli raz na jakiś czas nie sprawdzisz statusu zamówienia. Podobnie jak w poprzednim podejściu na Tobie jako na konsumencie usług spoczywa w tym przypadku odpowiedzialność za rekompensowanie niedociągnięć usługi.

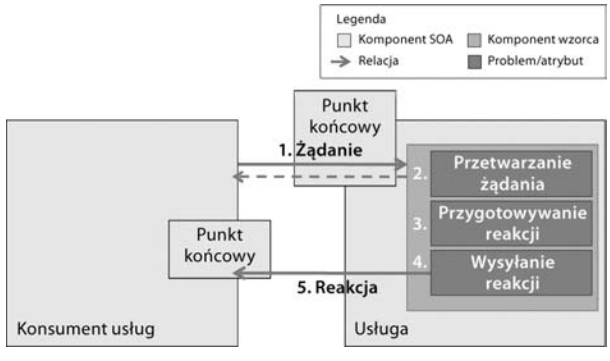
Istnieją rozwiązania w zakresie interakcji, które obsługują złożone interakcje. Należy do nich wzorzec Saga (omówiony w podrozdziale 5.4). Implementacja wzorca Saga rozwiąże ten problem, ale to jak strzelać z armaty do wróbli. Jest to nadmiar środków, kiedy potrzebujesz jedynie opóźnionej odpowiedzi.

ROZWIĄZANIE

Kiedy zastosowanie wzorca Saga jest przesadą, sprawdza się złamanie integracji. Uderza to jednak w konsumentów usług i lepiej unikać integracji po stronie klienta z powodu jej złych skutków. Tak naprawdę potrzebujesz jakoś zaimplementować komunikację asynchroniczną w SOA w możliwie jak najprostszy sposób. Oto co musisz zrobić:

✓ **Zastosuj wzorzec Żądanie/Reakcja i zaimplementuj komunikację asynchroniczną pomiędzy konsumentami usług a usługą. Zaimplementuj wymianę komunikatów w postaci dwóch komunikatów jednokierunkowych — żądanie ze strony konsumenta i odpowiedź ze strony usługi.**

Koncepcja wzorca Żądanie/Reakcja, przedstawionego na rysunku 5.5, zakłada posiadanie dwóch odrębnych interakcji pomiędzy konsumentem usług a usługą. Pierwsza interakcja wysyła żądanie do serwera, który może zwrócić potwierdzenie odbioru, bilet lub oszacować czas zakończenia zlecenia. Po zakończeniu przetwarzania usługa musi zainicjować interakcję z konsumentem usług i wysłać mu odpowiedź lub reakcję.



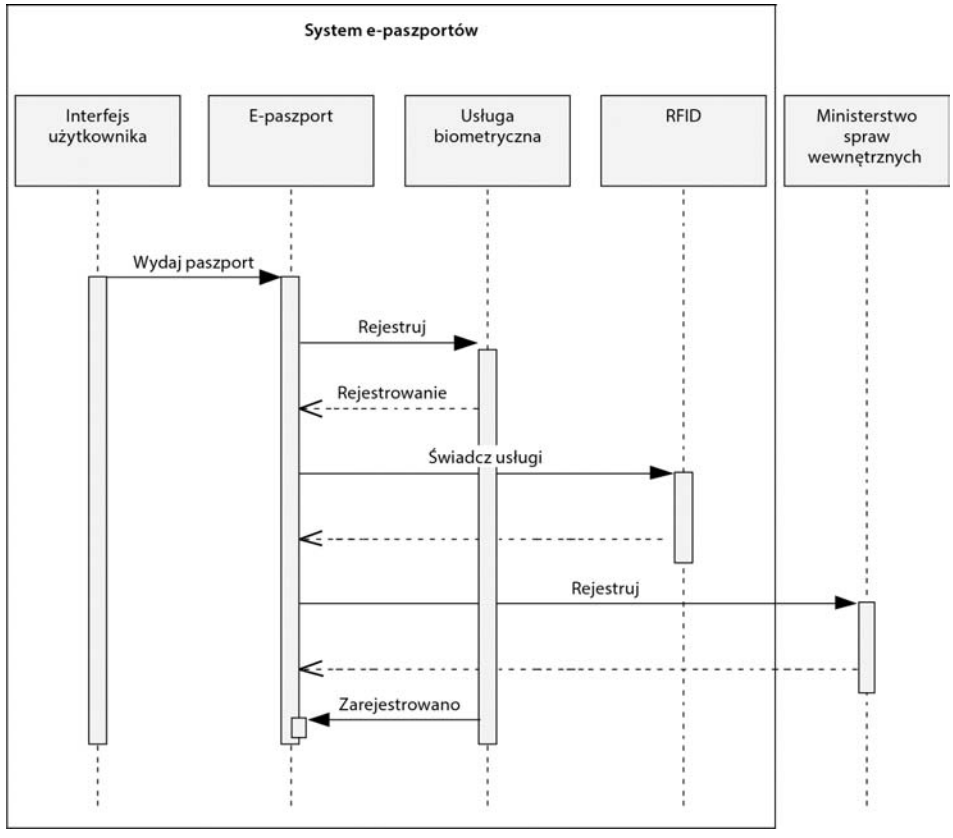
Rysunek 5.5. Wzorzec Żądanie/Odpowiedź definiuje komunikaty żądania i odpowiedzi w kontrakcie usługi. Kiedy usługa otrzymuje żądanie, przetwarza je i przygotowuje reakcję. Kiedy reakcja jest gotowa, usługa odsyła żądanie do konsumenta

UWAGA Usługa musi posiadać informacje, gdzie zwrócić odpowiedź — zajmiemy się tym później.

Wzorzec Żądanie/Reakcja jest bardziej zgodny z podstawową przesłanką wymiany komunikatów, ponieważ znosi powiązania czasowe. Dla porównania wzorzec Żądanie/Odpowiedź jest bardziej zgodny z RPC.

Na rysunku 5.6 przedstawiono zastosowanie wzorca Żądanie/Reakcja dla usługi biometrycznej. Teraz kiedy usługa biometryczna otrzymuje komunikat rejestracji, reaguje komunikatem „rejestrowanie" informującym klienta, że żądanie zostało odebrane. Po zakończeniu procesu rejestrowania, z powodzeniem lub z błędem, przygotowywana jest odpowiedź z rekordami rejestracji, która wysyłana jest do klienta.

UWAGA W scenariuszu przedstawionym na rysunku 5.6 sensowne byłoby zastosowanie wzorca Saga (omówionego w podrozdziale 5.4) do wycofywania pozostałych usług, jeśli sprawdzanie przez usługę biometryczną zakończyłoby się znalezieniem duplikatu tożsamości.



Rysunek 5.6. Proces wydawania paszportów wykorzystujący wzorzec Żądanie/Reakcja. Teraz usługa biometryczna zwraca dwa komunikaty. Najpierw zwraca potwierdzenie, że komunikat jest przetwarzany, a następnie po zakończeniu przetwarzania zwraca status

Wzorzec Żądanie/Reakcja jest wykorzystany we wzorcu Oddzielone Wywołanie (omówionym w rozdziale 2.). Różnica pomiędzy tymi dwoma wzorcami polega na tym, że Żądanie/Reakcja oddziela odpowiedź od żądania, podczas gdy Oddzielone Wywołanie oddziela także przetwarzanie komunikatu.

Semantyka interakcji wzorca Żądanie/Reakcja jest ograniczona. Jeśli scenariusz systemu e-paszportowego uwzględniałby możliwość anulowania rejestracji (np. w przypadku błędu świadczenia usług RFID), problematyczne byłoby skoordynowanie tego z grupą żądań i reakcji. W takich długotrwałych interakcjach można rozważyć zastosowanie bardziej zaawansowanych wzorców, takich jak wzorzec Saga opisany w podrozdziale 5.4.

Wzorzec Żądanie/Reakcja oferuje większą elastyczność niż Żądanie/Odpowiedź, ale ma to swoją cenę. Żądanie/Reakcja jest bardziej skomplikowanym wzorcem niż Żądanie/Odpowiedź i wymaga więcej pracy po stronie usługi (lub komponentu brzegowego).

Przyjrzyjmy się niektórym szczegółom implementacji, o które należy zadbać.

MAPOWANIE TECHNOLOGII

Najlepszym sposobem implementacji wzorca interakcji Żądanie/Reakcja jest zastosowanie dwóch jednokierunkowych komunikatów. Jeśli korzystasz z usług sieciowych, będzie to oznaczać dwa kanały HTTP. Jeśli stosujesz komunikaty, będziesz potrzebował kolejki (punktu końcowego) dla każdej z zainteresowanych stron.

Pierwszą przeszkodą jest czasowe oddzielenie. Ponieważ żądanie i reakcja (odpowiedź) są oddzielone w czasie, pozostałe komunikaty mogą znaleźć się pomiędzy nimi. Oznacza to, że musisz zapewnić sposób uzyskiwania przez usługę informacji, gdzie wysłać reakcję. Oznacza to także, że zarówno usługa, jak i konsument usług potrzebują sposobu korelacji komunikatów żądania i reakcji — więcej szczegółów na ten temat znajdziesz w ramce „Komunikaty skorelowane".

Zarówno Java, jak i .NET oferują rozwiązania z zakresu komunikatów jednokierunkowych. Biblioteka Javy Apache Axis2 oferuje nawet gotową infrastrukturę do implementacji kompletnego wzorca Żądanie/Reakcja. Listing 5.3 przedstawia kod po stronie klienta wymagany do wysyłania komunikatów asynchronicznych.

Z punktu widzenia architektonicznego reakcja jest komunikatem wysyłanym przez usługę. Jednak z punktu widzenia implementacji może ona być również zaimplementowana w formie pobierania jej przez konsumenta usług.

Komunikaty skorelowane

Jedno z wyzwań dotyczących asynchronicznej wymiany komunikatów wynika z faktu, że komunikat reakcji i żądanie nie są bezpośrednio powiązane. Reakcja może przychodzić długo po czasie wysłania pierwotnego żądania. W takiej sytuacji potrzebny jest sposób identyfikacji, że te dwa komunikaty są powiązane.

Mechanizm rozwiązujący ten problem znany jest jako identyfikator korelacji i jak wskazuje nazwa, polega on na dodaniu do komunikatów tokenu, który może być wykorzystany przez konsumentów usług i usługi do identyfikacji powiązanych komunikatów. Nie jest to bardzo odległe od koncepcji ciasteczek sesji w aplikacji WWW. Identyfikator korelacji może zawierać ID komunikatu, token konwersacji itd.

Korelacja jest obsługiwana przez szeroką gamę standardów WS-*. WS-Addressing posiada np. relację nagłówka identyfikatora komunikatu, która może być użyta dla określania korelacji. Kolejnym przykładem jest WS-BPEL, który ma nawet lepsze wsparcie dla korelacji, umożliwiając programistom definiowanie różnych zestawów korelacji wraz z zawartością tych zestawów.

Listing 5.3. Wykorzystujący wzorzec Żądanie/Reakcja kod klienta służący do wysyłania komunikatów

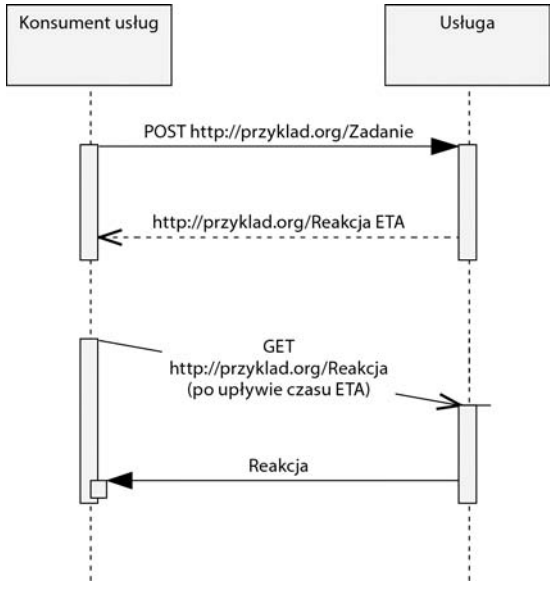
```
boolean useTwoChannels = true;

...
```

```
OMElement messageBody = helper.FormatmMessage(data,type);
Call msgSender = new Call();
msgSender.setTo(
        new EndpointReference(AddressingConstants.WSA_TO,
                "HTTP://www.example.org/ServiceName));
msgSender.setTransportInfo(Constants.TRANSPORT_HTTP,
    Constants.TRANSPORT_HTTP, useTwoChannels);
Callback callback = new Callback() {
        public void onComplete(AsyncResult result) {
         // tutaj należy wstawić kod do obsługi Reakcji
        }

        public void reportError(Exception e) {
         // kod obsługi błędów...
        }
};
msgSender.engageModule(new Qname("addressing"));
msgSender.invokeNonBlocking("MessageName", messageBody, callback);
```

Implementacja wzorca Żądanie/Reakcja na bazie wzorca Żądanie/Odpowiedź nie jest zbyt skomplikowana. Na rysunku 5.7 przedstawiono kolejne etapy. Kiedy konsument usług wysyła żądanie, otrzymuje w ramach odpowiedzi adres reakcji (w tym przypadku URI). Konsument otrzymuje również token czasu określający, kiedy można się spodziewać odpowiedzi. Po upływie wskazanego czasu konsument wysyła drugie żądanie do usługi, tym razem prosząc o odpowiedź (np. za pomocą polecenia `GET`).



Rysunek 5.7. Implementacja wzorca Żądanie/Reakcja na bazie wzorca Żądanie/Odpowiedź. Komunikat zwrotny żądania objaśnia, gdzie będzie można znaleźć reakcję i jaki jest szacowany czas dostarczenia (ang. *Estimated Time of Arrival — ETA*). Po upływie czasu ETA, jeśli konsument usług nie jest zajęty, może przejść do adresu reakcji w usłudze i samodzielnie pobrać reakcję

UWAGA Do śledzenia czasu możesz u konsumenta zastosować wzorzec Usługa Aktywna omówiony w rozdziale 2.

Tę drogę (technologię **push** zamiast **pull**) należy wybrać, kiedy nie jesteś w stanie stworzyć aktywnego, niezależnego punktu końcowego po stronie konsumenta. Ponownie preferowanym podejściem byłoby typowe zastosowanie wzorca Żądanie/Reakcja. Jednak jeśli nie możesz tego zrobić, to możesz zaimplementować podejście **pull** i nadal spełniać wymogi ogólnej koncepcji wzorca, które zakładają uzyskanie elastyczności i tymczasowego oddzielenia.

ATRYBUTY JAKOŚCIOWE

Wspomniałem, że tymczasowe oddzielenie i elastyczność zapewniane przez wzorzec Żądanie/Reakcja są głównymi atrybutami jakościowymi przemawiającymi za jego zastosowaniem. Wzorzec ten może być również pomocny w zakresie atrybutu jakościowego wydajności. Kiedy wysyłanie komunikatu do usługi nie blokuje konsumenta, pozwala to konsumentowi przydzielać cykle CPU do innych zadań (takich jak obsługa żądań z innych usług). Porównaj to z blokującym wzorcem Żądanie/Odpowiedź, który przytrzymuje zasoby po stronie konsumenta, oczekując na odpowiedź.

Tabela 5.3 przedstawia kilka przykładowych scenariuszy, w których wzorzec Żądanie/Reakcja ma większe zastosowanie niż inne wzorce.

Tabela 5.3. Atrybuty jakościowe wzorca Żądanie/Reakcja i przykładowe scenariusze

Atrybut jakościowy	Konkretny atrybut	Przykładowy scenariusz
Elastyczność	Tymczasowe powiązania	W normalnych warunkach system powinien powiadamiać stronę zamawiającą o wysyłce zamówienia w ciągu dwóch godzin od nadania paczki
Wydajność	Reagowanie	W normalnych warunkach interfejs użytkownika nie będzie zawieszony podczas wykonywania długotrwałych operacji (takich jak wyszukiwania i przeliczanie kursu)

Wzorzec Żądanie/Odpowiedź demonstruje synchroniczną komunikację pomiędzy konsumentami usług a usługami. Z kolei wzorzec Żądanie/Reakcja demonstruje komunikację asynchroniczną. Należy sprawdzić, kiedy możliwa jest komunikacja wykorzystująca architekturę sterowaną zdarzeniami bez naruszenia ograniczeń i założeń SOA.

5.3. Wzorzec Odwrócenie Komunikacji

Wzorce Żądanie/Odpowiedź i Żądanie/Reakcja nastawione są na interakcje, w których konsument chce uzyskać informację od usługi lub wymóc na niej jakieś działanie. W tym celu konsument gotów jest ponieść koszty powiązań, które są niezbędne do uzyskania informacji o usłudze, jej funkcjonalnościach oraz protokole (kontrakcie) stosowanym przez nią do udostępniania tych funkcjonalności.

Co się jednak stanie, kiedy potencjalny konsument nie będzie wiedział, że musi poprosić usługę o nowe informacje? Czy usługa sama go powiadomi? Czy usługa będzie chciała ponieść koszty powiązań?

Chociaż na pozór taka sytuacja może wydawać się mało prawdopodobna, przyjrzyjmy się kilku przykładom. Zobaczysz, że jest to dość typowa sytuacja biznesowa, która może być normą.

PROBLEM

Załóżmy, że chcesz stworzyć dla linii lotniczych usługę, która będzie proaktywnie zajmować się opóźnionymi lotami. Kiedy oczekiwane jest opóźnienie przylotu, możesz chcieć znaleźć nowe połączenia dla pasażerów, którzy nie zdążą na przesiadkę, zwolnić ich rezerwacje w bieżących odlotach oraz dostosować parametry dla tych lotów.

W tym celu musisz współpracować z kilkoma usługami — niektóre z nich będą częścią Twojego systemu (np. usługa śledząca wszystkie aktywne loty), a niektóre będą zewnętrzne (np. usługi dostarczające raporty pogodowe i informacje o stanach lotnisk). Na rysunku 5.8 pokazano informacje o opóźnieniach, które można uzyskać od Federalnej Administracji Lotnictwa (ang. *Federal Aviation Administration* — FAA) w Stanach Zjednoczonych.

INFORMACJA O STATUSIE LOTNISKA
dostarczona przez Centrum Dowodzenia Systemu Kontroli Ruchu Lotniczego FAA

Status czasu rzeczywistego dla lotniska Dallas/Ft Worth International Airport (DFW)

Zamieszczone na tej stronie informacje wskazują ogólne warunki portu lotniczego; nie dotyczą konkretnych lotów. Zasięgnij informacji u swojego przewoźnika, aby sprawdzić, czy Twój lot został opóźniony.

Opóźnienia według miejsca przeznaczenia: brak opóźnień.

Ogólne opóźnienia odlotów: opóźnienie odpraw i ruchu taksówkowego nieprzekraczające 15 minut.

Ogólne opóźnienia przylotów: opóźnienia w ruchu powietrznym nieprzekraczające 15 minut.

Ostatnia aktualizacja: **21.01.2007, 13:54 GMT+00:00**

Rysunek 5.8. Informacje o opóźnieniach przylotów i odlotów dostarczane przez FAA (http://www.fly.faa.gov/flyfaa/usmap.jsp). Może to być źródło informacji dla systemu kontroli ruchu lotniczego

Na rysunku 5.9 przedstawiono usługę „Opóźnienia" wraz z kilkoma usługami, z których ona korzysta.

UWAGA Jeśli poszukasz w internecie zdarzeń biznesowych, zauważysz, że przykład linii lotniczych jest całkiem popularny, ale jest też wiele innych bardziej praktycznych, sztampowych przykładów z dziedziny IT. Wyobraź sobie np. osobę, która chce wiedzieć, kiedy ceny akcji osiągną określony poziom, lub kogoś, kto chce być powiadamiany za każdym razem, kiedy składane jest zamówienie przekraczające określoną wartość. Podobnie system inwentaryzacyjny musi wiedzieć, że należy zamówić nową partię towaru, kiedy poziom zapasów spada poniżej określonego progu, a rozwiązania z zakresu dashboardingu i monitorowania aktywności biznesowej (ang. *business activity monitor* — BAM) muszą mieć informacje o problemach, które mają raportować.

Rysunek 5.9. Niektóre z usług, z którymi musiałaby współpracować usługa „Opóźnienia". Usługa „Opóźnienia" steruje kilkoma usługami bezpośrednio (takimi jak „Rezerwacje" i „Plany lotów"), ale sama jest sterowana danymi pochodzącymi z pozostałych usług („Pogoda", „Obraz operacyjny" i „Lotniska")

Chociaż architektura SOA wydaje się być zakorzeniona w interakcjach Żądanie/Odpowiedź, musisz również znaleźć sposób na obsługę zdarzeń biznesowych w ramach ograniczeń i dogmatów SOA. Innymi słowy:

? **Jak możesz obsłużyć zdarzenia biznesowe w SOA?**

Jedną z opcji jest trzymanie się podstawowego podejścia SOA i przygotowanie usługi, która generuje aktywnie zdarzenie i wysyła komunikat do wszystkich zainteresowanych usług. Zwróć uwagę, że dla takiego scenariusza usługa źródłowa musi mieć informacje o wszystkich zainteresowanych usługach, co obejmuje rozumienie ich kontraktów. Jest to problematyczne, ponieważ wprowadza niepotrzebne powiązania pomiędzy źródłem zdarzeń a pozostałymi usługami. W poprzednim przykładzie usługa „Pogoda" powinna mieć informacje o usługach „Opóźnienia" i „Obraz operacyjny". Analogicznie, jeśli usługa „Lotniska" potrzebowałaby informacji o pogodzie, aby aktualizować statusy lotnisk, musiałbyś wprowadzić zmiany w usłudze „Pogoda", żeby informowała również usługę „Lotniska". Musisz pamiętać, że w przeciwieństwie do klasycznego scenariusza Żądanie/Odpowiedź, nasza usługa źródłowa nie zajmuje się usługami docelowymi.

Inną opcją jest umożliwienie zainteresowanym usługom sondowania pod kątem aktualizacji. Każde zdarzenie zasadniczo posiada swój czas życia, kiedy jest wciąż dostępne w aktualnym stanie usługi będącej jego dostawcą. Zainteresowana usługa może sondować usługę generującą zdarzenia i dowiadywać się o interesujących zdarzeniach. Zaletą tego podejścia w stosunku do poprzedniej opcji jest to, że teraz kierunek zależności jest właściwy. Usługi przeprowadzające sondowanie są tymi, które są zainteresowane informacjami. Problem z sondowaniem polega na tym, że jeśli jego interwał jest zbyt długi, ominą Cię istotne zdarzenia. Z kolei jeśli interwał jest za krótki, wywołasz niepotrzebne obciążenie sieci. (Problem ten można rozwiązać — wrócę do tego w kontekście wariacji na temat danego rozwiązania).

Możesz załagodzić problem powiązań usług w opcji sondowania poprzez przesunięcie relacji na zewnątrz usług. Jednym ze sposobów na to jest wzorzec Orkiestracja (omówiony w rozdziale 7.), który wykorzystuje zewnętrzny silnik prze-

pływu pracy. Źródło zdarzeń może wtedy posiadać pojedynczą zależność w punkcie końcowym silnika przepływu pracy. Silnik przepływu pracy ma informacje o wszystkich zainteresowanych stronach i przekazuje im komunikaty.

Jest to krok w dobrym kierunku, ponieważ usługi nie są powiązane i łatwo jest wprowadzać zmiany do przepływu pracy, a także dodawać nowe usługi. Wadą jest stowarzyszenie logiki pomiędzy usługami a przepływem pracy.

Rozważyliśmy trzy różne rozwiązania, z których każde ma swoje zalety, ale może jesteśmy w stanie zaproponować coś lepszego? Myślę, że tak.

ROZWIĄZANIE

Rozwiązanie do obsługi zdarzeń biznesowych cały czas kryło się w tle. Jeśli chcesz dodawać zdarzenia, to czemu by nie zaadaptować stylu architektonicznego zbudowanego wokół zdarzeń i nie wcielić tego do SOA? Tak się składa, że nie musimy ponownie wymyślać koła — istnieje już odpowiedni styl architektoniczny, zwany architekturą sterowaną zdarzeniami (ang. *event-driven architecture* — EDA).

Zdarzenie (ang. *event*) jest znaczącą zmianą mającą miejsce wewnątrz generatora zdarzeń lub komponentu, który jest obserwowany przez generator zdarzeń. Specyfikacje zdarzeń w EDA to zorganizowane encje podobne do kontraktów i komunikatów SOA. Specyfikacja zdarzenia składa się z nagłówka i treści, gdzie nagłówek zawiera metadane, a treść zawiera faktyczną informację na temat zdarzenia. W przeciwieństwie do tradycyjnych komunikatów, zdarzenia nie posiadają określonego miejsca przeznaczenia.

EDA jest zbliżone do wzorca publikuj/subskrybuj (ang. *publish/subscribe*), ale posiada także kilka różnic, takich jak perspektywa historyczna, która jest uzyskana poprzez traktowanie zdarzeń jako strumieni, a nie wyizolowanych wystąpień.

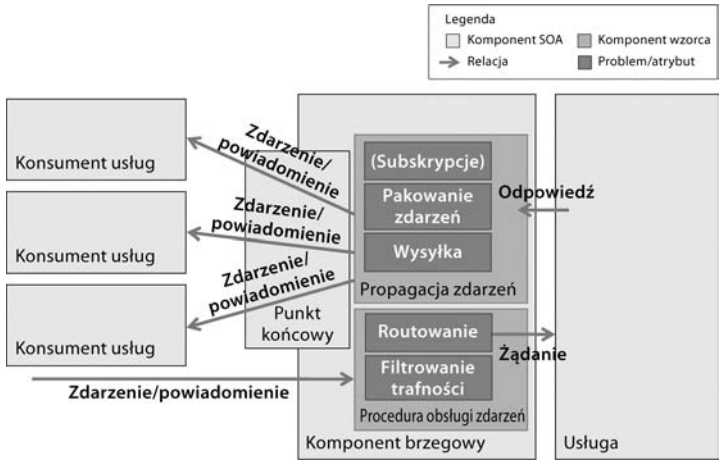
Aby dostosować do SOA zdarzeniowy wzorzec wymiany komunikatów, możesz zrobić, co następuje:

✓ **Zaimplementuj wzorzec Odwrócenie Komunikacji, uzupełniając SOA o architekturę EDA — możesz umożliwić usługom publikowanie strumienia zdarzeń, które w nich występują, zamiast bezpośrednio wywoływać usługi.**

Przedstawiony na rysunku 5.10 wzorzec Odwrócenie Komunikacji zasadniczo zmienia kierunek przepływu informacji. Zamiast wywoływania usług przez konsumentów usług w celu uzyskania informacji, usługi docierają do konsumentów z aktualizacjami. Ta zamiana ról wymaga dwóch komponentów w obrębie usługi lub raczej w komponencie brzegowym (ponieważ komponenty te nie są zorientowane biznesowo).

Pierwszym komponentem jest propagacja zdarzeń. Zdarzenia powinny być pakowane do formatu uzgodnionego dla inicjatywy SOA (lub zgodnie z kontraktem usługi, jeśli nie ma wspólnego kontraktu) i dystrybuowane (omawia to kolejny punkt, poświęcony mapowaniu technologii).

Drugi komponent, procedura obsługi zdarzeń, umożliwia usłudze działanie jako konsument usług dla zdarzeń wysłanych przez inne usługi. Pierwszym zadaniem procedury obsługi zdarzeń jest filtrowanie przychodzących zdarzeń pod kątem

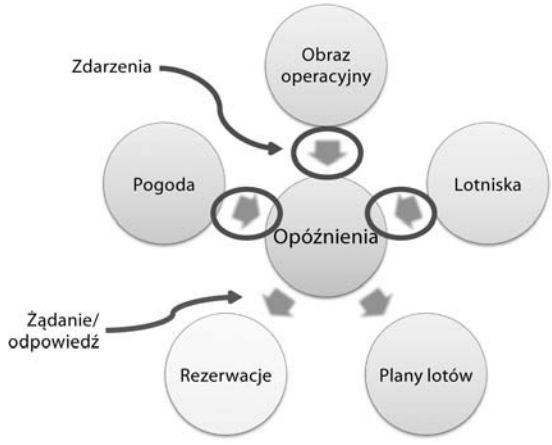


Rysunek 5.10. We wzorcu Odwrócenie Komunikacji komponent brzegowy usługi poza „standardowymi" żądaniami akceptuje i filtruje przychodzące zdarzenia. Kiedy usługa ma gotową jakąś odpowiedź lub reakcję na zdarzenie, komponent brzegowy również pakuje i ekspediuje ją do konsumentów usług jako zdarzenie

trafności. Jest to istotne, ponieważ wiele odebranych zdarzeń może nie być istotne, szczególnie jeśli infrastruktura pomiędzy usługami nie jest wystarczająco inteligentna, by routować subskrypcje lub nimi zarządzać. Drugą rolą procedury obsługi zdarzeń jest routowanie odpowiednich zdarzeń do komponentów usługi, które mogą reagować na te zdarzenia — komponenty, które w modelu Żądanie/Odpowiedź otrzymałyby nowe informacje w postaci żądań.

Jak pewnie zauważyłeś, chociaż wzorzec Odwrócenie Komunikacji mówi o zdarzeniach, to nie posiada w komponencie brzegowym komponentu zarządzania subskrypcjami. Dzieje się tak dlatego, że zarządzanie subskrypcjami wymaga zbyt wiele wysiłku niezwiązanego tak naprawdę z usługami, takiego jak routowanie i subskrypcje trwałe. Alternatywą do subskrypcji w usługach jest przeniesienie odpowiedzialności na konsumenta lub infrastrukturę (lub oba te składniki). W tym celu możesz dostarczyć znane nazwy (adresy URI, kolejki itp.), pod którymi można znaleźć zdarzenia, a następnie skonfigurować zainteresowane usługi do ich nasłuchiwania.

Przyjrzyjmy się usłudze „Opóźnienia" wspomnianej w opisie problemu. Na rysunku 5.11 widać, że teraz usługi „Lotniska", „Pogoda" i „Obraz operacyjny" przekazują swoje zmiany usłudze „Opóźnienia", a nie na odwrót. Ma to pozytywny wpływ na ruch sieciowy, ponieważ usługa „Opóźnienia" nie musi się dłużej przejmować tym, że pominie jakąś istotną zmianę w trzech usługach, które monitoruje. Zwróć też uwagę, że zastosowanie wzorca Odwrócenie Komunikacji nie oznacza, że musisz przenosić wszystkie swoje interakcje na zdarzenia. W tym przykładzie usługa „Opóźnienia" wciąż posiada interakcje Żądanie/Odpowiedź z usługami „Plany lotów" i „Rezerwacje". Jeśli usługa „Opóźnienia" zidentyfikuje opóźnienie, może spróbować zarezerwować miejsca w późniejszych lotach dla osób, które nie zdążyły na przesiadkę.



Rysunek 5.11. Relacje pomiędzy usługami z rysunku 5.9 po zastosowaniu wzorca Odwrócenie Komunikacji. Teraz usługi „Pogoda", „Lotniska" i „Obraz operacyjny" przekazują swoje zmiany do usługi „Opóźnienia"

Należy zwrócić uwagę, że przy łączeniu wzorca Odwrócenie Komunikacji z wzorcem Żądanie/Reakcja lub Żądanie/Odpowiedź poza odpowiadaniem konsumentowi usługi (lub jako alternatywa tej czynności) usługa powinna również zgłosić zdarzenie, informując elementy nasłuchujące o efektach obsługi żądania, tak aby subskrybowane usługi mogły obsłużyć efekty zmian.

We wzorcu Odwrócenie Komunikacji chodzi o implementację EDA na bazie SOA. Jak dotąd przyglądaliśmy się prostej stronie tej kwestii, która obejmuje sporadyczne lub wyizolowane zdarzenia. Ale bardzo silna koncepcja definiowana przez EDA to **strumień zdarzeń**. Oznacza to, że nie spoglądasz na zdarzenie w jego własnej postaci, ale raczej na łańcuch powiązanych zdarzeń w pewnym przedziale czasowym. Strumienie zdarzeń mogą dostarczyć Ci zarówno trendy, jak i perspektywę historyczną. Dobrze wykorzystane mogą zapewnić Ci wywiad biznesowy i monitorowanie aktywności biznesowej w czasie rzeczywistym. Wzorzec Zagregowane Raportowanie, omówiony w rozdziale 7., pokazuje zastosowanie tych funkcjonalności.

Kolejnym wzorcem, który możesz połączyć z Odwróceniem Komunikacji, jest wzorzec Potoki Równoległe (omówiony w rozdziale 3.). Ta kombinacja może być zastosowana do dostarczenia implementacji SOA etapowej architektury sterowanej zdarzeniami (ang. *staged event-driven architecture* — SEDA). W skrócie, SEDA może zapewnić możliwość zwiększenia współbieżności i przepustowości rozwiązania w stosunkowo prosty sposób.

Wadą zastosowania wzorca Odwrócenie Komunikacji jest dodatkowa złożoność projektowania całego systemu wykorzystującego zdarzenia. Sposób radzenia sobie z tym problemem został już wspomniany — nie stosuj wyłącznie wzorca Odwrócenie Komunikacji, a raczej połącz go z innym wzorcem wymiany komunikatów wymienionym w tym rozdziale.

Jedną z rzeczy, na które należy uważać i których należy unikać przy korzystaniu z wzorca Odwrócenie Komunikacji, jest błędne koło zdarzeń. Ma to miejsce, kiedy jakieś zdarzenie uruchamia łańcuch zdarzeń, który wraca do źródła pierwotnego zdarzenia i powoduje ponowne uruchomienie tego samego lub podobnego zdarzenia.

Nie spotkałem się jeszcze z tym w praktyce, ale taka możliwość istnieje. Sposobem radzenia sobie z tym problemem jest rejestrowanie i monitorowanie, np. za pomocą wzorca Strażnik Usługi omówionego w rozdziale 3.

Przestawienie się na wzorzec Odwrócenie Komunikacji komplikuje również procesy debugowania. Gdy coś pójdzie nie tak, musisz prześledzić problem wstecz, aż do motyla, którego skrzydła zainicjowały reakcję łańcuchową prowadzącą do tego problemu. Sposobem przeciwdziałania jest scentralizowane rejestrowanie przez cały proces rozwoju (i prawdopodobnie również w fazie produkcyjnej), co umożliwia odtworzenie systemu. Jest to bardziej skomplikowane niż podążanie za bezpośrednim wywołaniem stosu.

Kolejnym wyzwaniem przejścia na wzorzec Odwrócenie Komunikacji jest dodanie go, kiedy jesteś w trakcie budowania inicjatywy SOA i posiadasz już wdrożone usługi wykorzystujące prostsze wzorce wymiany komunikatów. Nie potrafię dostarczyć ogólnych wskazówek dotyczących remodelowania interakcji, ponieważ jest to ściśle uzależnione od sytuacji. Jeśli jednak chodzi o samą inicjatywę SOA, sekretem jest stopniowe przechodzenie.

Inny zestaw wyzwań związanych z wzorcem Odwrócenie Komunikacji dotyczy szczegółów implementacji. Wszakże wiele infrastruktur SOA (co najbardziej oczywiste, HTTP) nie obsługuje zdarzeń czy transmisji grupowych (multicastów). Zobaczmy, czy uda nam się przezwyciężyć te przeszkody.

MAPOWANIE TECHNOLOGII

Istnieje kilka opcji mapowania technologii dla implementacji wzorca Odwrócenie Komunikacji.

Pierwszą opcją, która wydaje się również najbardziej naturalna, jest zastosowanie korporacyjnej magistrali usług (ESB). Większość implementacji ESB może zaadaptować wszystkie popularne wzorce wymiany komunikatów, w tym publikuj/subskrybuj. Listing 5.4 pokazuje, jak można skonfigurować subskrypcję w Apache ServiceMix (ESB na licencji open source). W tym celu dodaj sekcję subskrypcji (`sm:subscription`) w sekcji konfiguracyjnej komponentu (`sm:activationSpecs`).

Listing 5.4. Fragment kodu konfiguracyjnego dołączający subskrypcję komponentu „picture”

```
<sm:activationSpecs>
    <sm:activationSpec componentName="sub" service="foo:Subscriber">
...
        <sm:subscriptions>
            <sm:subscriptionSpec service="cop::picture"/>
        </sm:subscriptions>
    </sm:activationSpec>
</sm:activationSpecs>
```

Kiedy chcesz zaimplementować wzorzec Odwrócenie Komunikacji z ESB, delegujesz odpowiedzialność za przekazywanie zdarzeń i zarządzanie subskrypcjami na infrastrukturę i wtedy możesz skoncentrować się na planowaniu zdarzeń oraz innych aktywnościach biznesowych.

Możesz uzyskać jeszcze luźniejsze powiązania, stosując infrastrukturę wymiany komunikatów (lub ESB), która obsługuje tematy, choć nie jest to typowa infrastruktura usług dla SOA. Tematy są luźniej powiązane, ponieważ subskrybenci nie wiedzą, kim jest wydawca — mają informacje jedynie o temacie, który ich interesuje. Problemem tego podejścia jest to, że skoro subskrybenci nie wiedzą, kim jest wydawca, to infrastruktura musi zadbać o to, żeby zdarzenia były publikowane tylko przez uwierzytelnione i zautoryzowane usługi.

Teraz zajmijmy się bardziej problematycznymi infrastrukturami, takimi jak HTTP (usługi RESTful) oraz proste TCP. Mamy tutaj dwie opcje.

Pierwsza opcja to napisać niezbędną infrastrukturę jako element komponentu brzegowego każdej usługi. Innymi słowy, należy przygotować własną logikę utrzymywania subskrypcji i aktywnego wysyłania każdego wygenerowanego zdarzenia do wszystkich zainteresowanych subskrybentów. Chociaż jest to technicznie wykonalne, nie zalecam podążania tą drogą, chyba że jesteś dostawcą platform pośredniczących. Lepiej skupić się na podstawowej działalności i wartości biznesowej swoich rozwiązań i nie próbować tworzyć delikatnego elementu infrastruktury, co i tak prawdopodobnie nie wyjdzie za pierwszym razem.

Druga opcja, według mnie bardziej interesująca, związana jest z aplikacjami typu *push* (tak właściwie to *pull*), których prawdopodobnie używasz na co dzień — blogi i ich czytelnicy. Kiedy publikuję na blogu nowe zdarzenie (post), nie jest ono natychmiastowo wysyłane do subskrybentów blogu. Właściwie nigdy nie jest aktywnie wysyłane. Zamiast tego nowe zdarzenie jest dodawane do strumienia zdarzeń (kanał RSS lub Atom), który zawiera najnowsze wydarzenia. Subskrybenci, którzy zarządzają subskrypcją po swojej stronie niezależnie ode mnie (luźne powiązania), decydują, jak często muszą sondować mój strumień zdarzeń, żeby nie przegapić istotnych zdarzeń. Ta decyzja jest oparta na liczbie zdarzeń utrzymywanych w strumieniu, częstotliwości nowych zdarzeń oraz opóźnieniu, na jakie subskrybenci mogą sobie pozwolić przy obsłudze zdarzeń. Zwróć uwagę, że konsumenci, którzy potrzebują małego opóźnienia pomiędzy wystąpieniem zdarzenia a powiadomieniem, będą prawdopodobnie potrzebować powiadamiania online i nie będą mogli skorzystać z tej metody.

Jak pewnie zauważyłeś podczas omawiania wzorca Żądanie/Odpowiedź (podrozdział 5.1), protokół APP (ang. *Atom Publishing Protocol*) jest popularnym wyborem dla formalizowania kolekcji w usługach sieciowych typu RESTful, podobnie jak wersje formatu JSON, takie jak OData i GData.

Jak wcześniej wspomniano, element architektury EDA we wzorcu Odwrócenie Komunikacji pozwala traktować zdarzenia jako strumień, a nie jako wyizolowane instancje. Strumienie zdarzeń mogą poprawić Twoje rozwiązania jeszcze bardziej, jeśli zastosujesz dodatkową koncepcję architektoniczną zwaną złożonym przetwarzaniem zdarzeń (ang. *complex event processing* — CEP). Jak wskazuje nazwa, CEP polega na badaniu strumieni zdarzeń pod kątem złożonych wzorców. Prawdopodobnie najłatwiej będzie wyjaśnić to na przykładzie.

Czas życia zdarzenia

Bez względu na to, czy do publikowania zdarzeń wykorzystujesz kanały (ang. *feeds*), czy też stosujesz podejście oparte na kolejkach, musisz wziąć pod uwagę czas życia (ang. *time to live* — TTL) zdarzeń. Poprzez TTL rozumiem przedział czasu, w którym zdarzenie powinno być dostępne dla konsumentów, zanim stanie się nieistotne.

Jeśli wykorzystujesz zdarzenia w języku programowania, parametr TTL jest nieodłączny („wszystkiemu trzeba poświęcić należytą uwagę, jeśli chce się uniknąć porażki"). Jeśli konsument jest nieobecny, kiedy zdarzenie jest przywoływane, to jego problem. W SOA rozsądniej jest pozwolić na tymczasowe oddzielenie pomiędzy czasem wywołania zdarzenia a czasem jego skonsumowania. Takie tymczasowe oddzielenie umożliwia zwiększenie autonomii i zapewnia luźne powiązania zarówno dla generatora zdarzeń, jak i konsumenta zdarzeń. Druga strona medalu jest taka, że teraz musisz uwzględniać czas życia zdarzeń, aby uniknąć przetwarzania przestarzałych informacji, zbyt dużego opóźnienia oraz problemów z wydajnością.

TTL zmienia się w zależności od znaczenia biznesowego zdarzenia, nie ma więc żadnych sztywnych reguł. Dwie ogólne zasady to takie, że TTL dla zdarzeń cyklicznych (np. aktualizacje cen akcji) jest zazwyczaj częstotliwością cyklu, a TTL dla zdarzeń jednorazowych (np. nowe zamówienie) jest z reguły znacznie dłuższe.

Listing 5.5 pokazuje przykładowe zapytanie we wbudowanym silniku CEP, które napisałem kilka lat temu (oparte na technologii C# LINQ). Zapytanie to analizuje strumień zdarzeń logowania i uruchamia alarm za każdym razem, kiedy wystąpią trzy nieudane logowania pod rząd u tego samego użytkownika.

Listing 5.5. Ustawiczne zapytanie dotyczące wywołania określonego zdarzenia w przypadku trzech kolejnych nieudanych logowań

```
var loginRecords = engine.GetEventSource<Login>();

engine.AddQuery(() => from names in loginRecords.Stream
                      group names by names.Name
                      into logins
                      from login in logins
                      let next = logins.FirstOrDefault(
t => t.LoginTime > login.LoginTime)

                      let nextNext = null == next ? null
                      ↪: logins.FirstOrDefault(t => t.LoginTime > next.LoginTime)
                      where
                        !login.Successful && (null != next && !next.Successful) &&
                        ↪(null != nextNext && !nextNext.Successful)
                        select login, HandleAlert);
```

Istnieje wiele komercyjnych silników CEP firm takich jak SAP, TIBCO czy IBM. Dostępnych jest też kilka opcji na licencji open source, np. Esper firmy EsperTech.

Wzorzec Odwrócenie Komunikacji stanowi dobrą okazję do wprowadzenia CEP do projektu, ale nie jest to główny powód korzystania z tego wzorca. Jak zwykle zakończymy naszą dyskusję o wzorcu, badając niektóre motywacje do jego zastosowania.

ATRYBUTY JAKOŚCIOWE

Odwrócenie Komunikacji to wzorzec o dużych możliwościach. Interakcja oparta na znaczeniach pomaga znacząco zwiększyć autonomię i kompozycyjność systemu, a także podnieść poziom ponownego wykorzystywania elementów wewnątrz systemu. To dla SOA świetna nowina do tego stopnia, że Gartner nazwał połączenie EDA z SOA „zaawansowanym SOA". Chociaż trzeba mieć na uwadze wyzwania związane z implementacją wzorca Odwrócenie Komunikacji, takie jak skomplikowane debugowanie i dodatkowa praca przy projektowaniu zdarzeń, jest to istotny wzorzec, który warto mieć w zestawie swoich narzędzi z uwagi na wszystkie oferowane przez niego korzyści.

Tabela 5.4 wskazuje niektóre scenariusze, które mogą skłonić Cię do zastosowania wzorca Odwrócenie Komunikacji.

Tabela 5.4. Atrybuty jakościowe wzorca Odwrócenie Komunikacji i przykładowe scenariusze

Atrybut jakościowy	Konkretny atrybut	Przykładowy scenariusz
Elastyczność	Oddzielanie	Usługi powinny posiadać możliwie najmniej informacji o sobie
Ponowne wykorzystanie	Interfejsy	Wszystkie usługi, poza konkretnymi żądaniami, które mogą obsługiwać, powinny obsługiwać kilka typowych interfejsów API
Łatwość wprowadzania zmian	Dodawanie funkcji	Posiadając gotową nową funkcjonalność, powinieneś być w stanie zintegrować ją w systemie w czasie krótszym niż trzy tygodnie

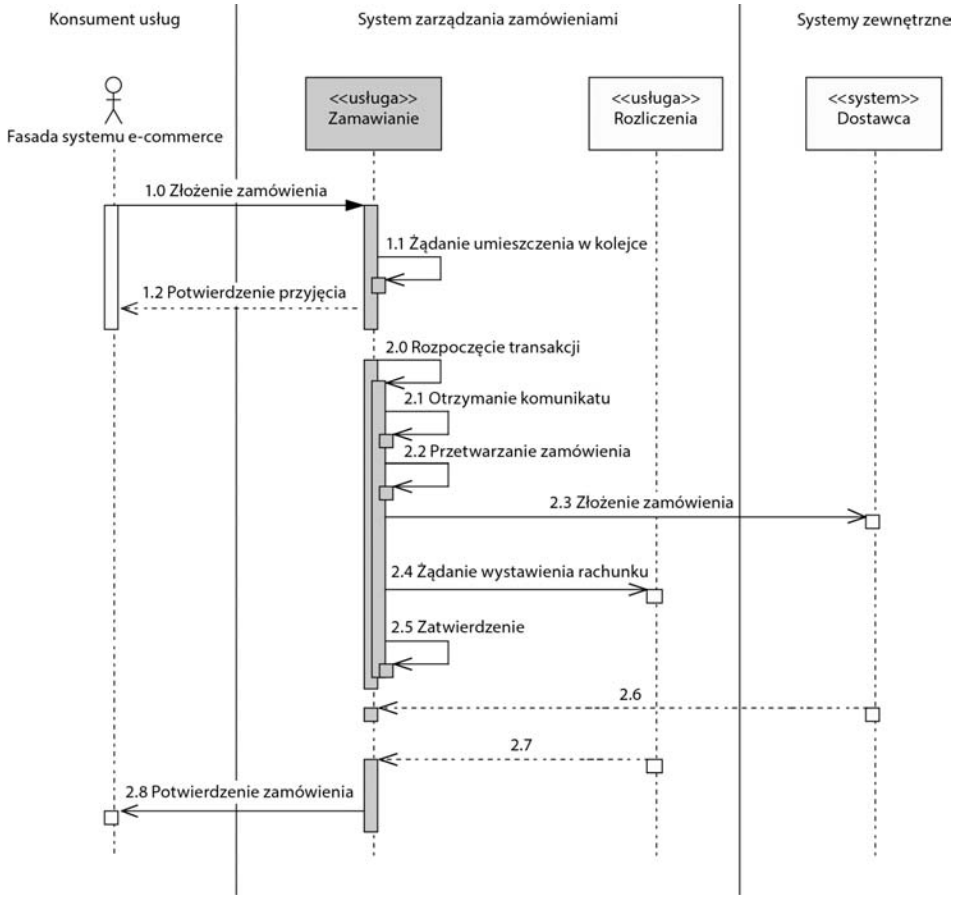
Wzorzec Odwrócenie Komunikacji podsumowuje podstawowe wzorce wymiany komunikatów, pokazując, w jaki sposób można w SOA wykorzystać zdarzenia lub model publikuj/subskrybuj. Ostatni z wzorców, którym zajmiemy się w tym rozdziale, nosi nazwę Saga. Umożliwia on uzyskanie transakcyjnego zachowania pomiędzy usługami.

5.4. Wzorzec Saga

W rozdziale 2. omawialiśmy wzorzec Usługa Transakcyjna, który zapewnia niezawodność obsługi żądań przez usługi. Jednak zastosowanie tego wzorca rozwiązuje tylko jedną część układanki. Przyjrzyjmy się ponownie scenariuszowi z rozdziału 2. i zobaczmy, co jeszcze powinniśmy zrobić.

Na rysunku 5.12 przedstawiona została usługa „Zamawianie", przetwarzająca zamówienie. Interesujące kwestie pojawiają się tutaj na etapach 2.3 i 2.4. W ramach wewnętrznej transakcji obsługującej żądanie usługa „Zamawianie" musi współpracować z dwoma innymi usługami: żąda wystawienia rachunku od wewnętrznej usługi „Rozliczenia" i składa zamówienia (na części lub materiały) w zewnętrznym systemie „Dostawca".

Napotkamy tutaj dwa główne problemy. Pomyśl, co się stanie, kiedy zamiast zatwierdzenia wewnętrznej transakcji na etapie 2.5 usługa „Zamawianie" zdecyduje



Rysunek 5.12. Przykładowy schemat przepływu komunikatów w scenariuszu e-commerce (konwersującego z usługą „Zamawianie"). Fasada wysyła zamówienie do usługi zamawiania, która następnie zamawia części w usłudze „Dostawca" i prosi usługę „Rozliczenia" o wystawienie rachunku klientowi. Zwróć uwagę, że cała obsługa komunikatu „Złożenie zamówienia" (etap 1.0) odbywa się w pojedynczej transakcji lokalnej (etapy od 2.0 do 2.5)

się przerwać swoją (wewnętrzną) transakcję. Zastanów się również, w jaki sposób usługa zamawiania mogłaby uzyskać zaangażowanie pozostałych usług, aby kontynuować swoją pracę na bazie tego zaangażowania. Zanim potwierdzisz zamówienie klientowi, możesz chcieć uzyskać potwierdzenie od dostawcy, że zamówione elementy zostały zarezerwowane dla Ciebie.

PROBLEM

Oczywistym rozwiązaniem tych dwóch wspomnianych w poprzednim punkcie problemów jest rozszerzenie wewnętrznej transakcji usługi „Zamawianie" na pozostałe usługi. Taka rozszerzona transakcja określana jest mianem **transakcji rozproszonej** (ang. *distributed transaction*).

Wykorzystując transakcje rozproszone, usługa „Zamawianie" musiałaby wywoływać zarówno usługę „Rozliczenia", jak i system „Dostawca" w ramach pojedynczej transakcji. Jeśli wszystkie usługi zgodziłyby się zaangażować, cała transakcja byłaby zatwierdzana i wypełniana jednocześnie. Brzmi to naprawdę świetnie i mamy nawet technologię, która to umożliwia — technologię wyprzedającą SOA o wiele lat.

Zawsze jednak istnieje jakieś „ale". Co jeśli dostawca może zakończyć swoją część transakcji dopiero po zatwierdzeniu biznesu przez starszego menedżera? Czy możesz przytrzymać swoje wewnętrzne blokady, czekając, aż ten menedżer wróci z wakacji na Bahamach w przyszłym tygodniu? Prawdopodobnie nie. A co jeśli dostawca okaże się być konkurencją? Mógłby przedłużać transakcję, aby zepsuć Twój biznes — oczekując zakończenia transakcji przez dostawcę, przytrzymujesz blokady na wewnętrznych zasobach.

Ten konkretny scenariusz może być zbyt daleko idący, ale chodzi o to, że nie możesz przyjmować założeń dotyczących sposobu działania pozostałych usług. Dotyczy to szczególnie usług, które nie należą do Ciebie. O innych powodach unikania transakcji wielousługowych możesz poczytać w kontekście antywzorca Integracja Transakcyjna w rozdziale 8.

Nawet jeśli sądzisz, że transakcje wielousługowe nie są problematyczne jako koncepcja, prawdopodobnie i tak zgodzisz się, że transakcje długotrwałe nie są zbyt dobre. Im bardziej konwersacyjna staje się transakcja pomiędzy usługami, tym bardziej powinieneś myśleć o alternatywach dla transakcji niepodzielnych. Na rysunku 5.12 widać dwa komunikaty wychodzące z usługi „Zamawianie", co może być granicą dla liczby interakcji. Jednak procesy biznesowe mogą czasem obejmować znacznie bardziej szczegółowe konwersacje.

Duża liczba komunikatów przepływających pomiędzy usługami nie jest zalecana, ponieważ zwiększa opóźnienia i możliwość wystąpienia błędu. Niemniej jednak mała liczba interakcji jest mało prawdopodobna. Usługi rzadko egzystują w całkowitej izolacji. Interoperacyjność jest jednym z pierwotnych powodów stosowania SOA. Oznacza to, że musisz obsłużyć w sposób niezawodny złożone interakcje usług bez łączenia wszystkiego w jedną długą transakcję niepodzielną.

Podsumujmy teraz nasze problemy.

? **Jak możesz osiągnąć rozproszony konsensus pomiędzy usługami bez zastosowania transakcji?**

Myślę, że stało się już jasne, iż zastosowanie pojedynczej transakcji nie wchodzi w grę. Jeśli wszystkie zaangażowane usługi są pod Twoją kontrolą, możesz rozbić długotrwały proces na kilka etapów, z których każdy będzie odbywał się w osobnej transakcji. Mniejsze rozproszone transakcje są z pewnością krokiem w dobrym kierunku, ale wciąż jesteś związany transakcjami wielousługowymi. Ponieważ wszystko nie jest ograniczone do pojedynczej transakcji, masz problemy, takie jak wycofanie efektu pierwszego etapu, jeśli coś pójdzie nie tak na trzecim czy czwartym etapie.

Inną opcją jest takie modelowanie kontraktów, żebyś nigdy nie potrzebował tego rodzaju złożonych interakcji. Możesz zredukować interakcje pomiędzy usługami,

jeśli zwiększysz ziarnistość tych usług. Istnieje jednak ograniczenie co do wielkości usług — nie chcesz przecież skończyć z jedną monolityczną usługą, która zajmuje się wszystkim. Ponadto, podobnie jak obiekty, usługi muszą być spójne i stosować się do zasady jednej odpowiedzialności. Jeśli tak zrobisz, możesz zawrzeć nieco interakcji w granicach usługi, ale nadal będziesz musiał obsłużyć interakcje wielousługowe, aby zaimplementować procesy biznesowe.

Pozostaje Ci więc opcja podzielenia interakcji usług — procesu biznesowego — na kilka mniejszych etapów i wymodelowanie ich tak, aby powstała długotrwała konwersacja pomiędzy usługami.

ROZWIĄZANIE

Wzorzec interakcji Saga ma zapewniać semantykę i komponenty do obsługi długotrwałych konwersacji wspomnianych pod koniec poprzedniego punktu.

✓ **Zaimplementuj wzorzec Saga i podziel interakcje usług (proces biznesowy) na kilka mniejszych akcji i reakcji biznesowych. Koordynuj konwersację i zarządzaj nią poprzez komunikaty i limity czasu.**

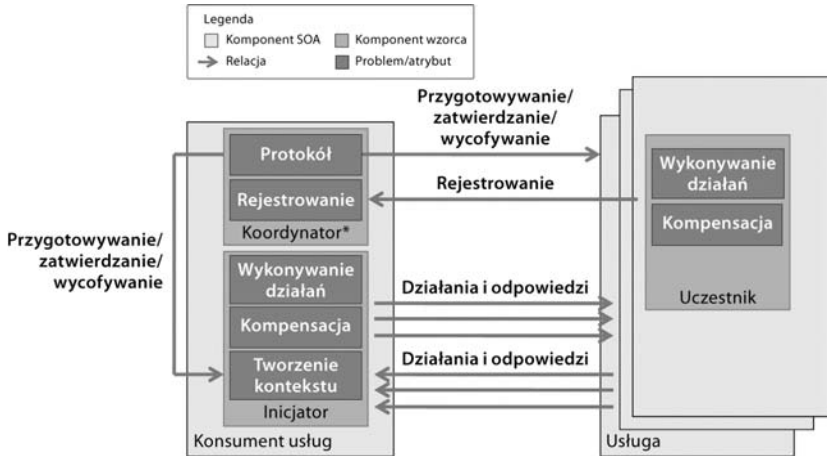
Hector Garcia-Molina i Kenneth Salem w 1987 r. zdefiniowali pojęcie „saga" jako sposób rozwiązania problemu długo żyjących transakcji bazy danych. Opisali oni sagę jako sekwencję powiązanych, niewielkich transakcji[1]. W sadze koordynator (w ich przypadku baza danych) zapewnia zakończenie z powodzeniem wszystkich zaangażowanych transakcji. W przeciwnym razie, jeśli transakcje się nie powiodą, koordynator uruchamia transakcje kompensujące, aby skorygować częściowe wykonanie.

To, co sprawdzało się w przypadku baz danych, jeszcze lepiej sprawdza się dla interakcji usług w SOA. Na rysunku 5.13 przedstawiono sposób, w jaki można zastosować koncepcję sagi w SOA. Możesz podzielić długotrwałą interakcję między usługami na poszczególne działania lub czynności oraz kompensacje (w przypadku awarii lub błędów).

Pierwszym komponentem z rysunku 5.13, o którym należy wspomnieć, jest inicjator. Inicjator uruchamia wzorzec Saga, tworząc kontekst, który jest powodem zaistnienia interakcji. Następnie inicjator wysyła zapytanie do jednej lub kilku pozostałych usług (uczestników) o wykonanie pewnych działań biznesowych. Uczestnik może się zarejestrować do koordynacji (w zależności od stopnia formalności implementacji Sagi). Uczestnicy i inicjator wymieniają komunikaty oraz żądania do momentu, aż osiągną pewne porozumienie lub będą gotowi do zakończenia interakcji. Wtedy koordynator wysyła żądania do wszystkich uczestników (również do inicjatora), aby sfinalizowali porozumienie (przygotowali się do zatwierdzenia) i zatwierdzili je.

Jeśli w trakcie interakcji lub fazy finalnej wystąpi jakiś problem, działania, które miały miejsce, muszą zostać anulowane. W regularnych transakcjach ACID można to zrobić, ale w sadze musisz przeprowadzić czynności korygujące zwane **kompen-**

[1] Hector Garcia-Molina i Kenneth Salem, „Sagas", w *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (1987), 249–59.

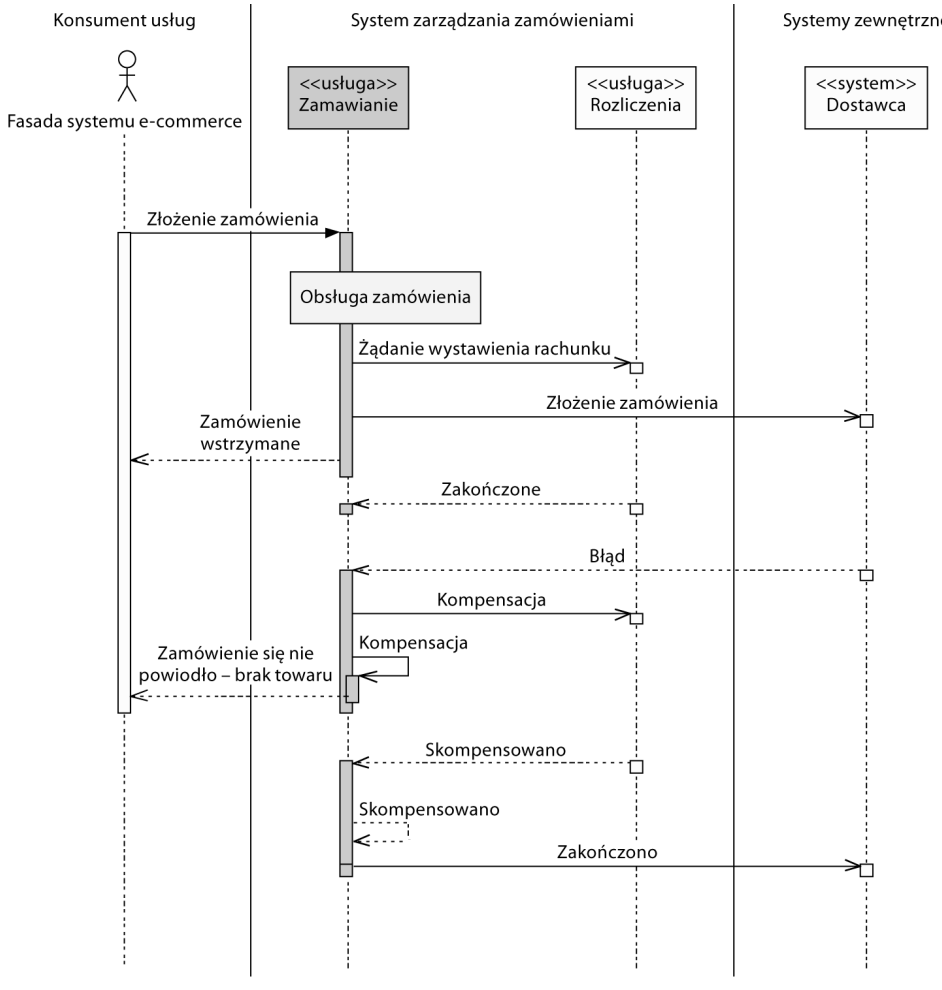


Rysunek 5.13. We wzorcu Saga konsument usługi oraz jedna lub kilka usług utrzymują długotrwałą konwersację w obrębie pojedynczego kontekstu (sagi). Kiedy strony osiągną pewien konsensus, konwersacja jest zatwierdzana. Jeśli w trakcie konwersacji wystąpią jakieś problemy, interakcja jest przerywana, a zaangażowane strony wykonują czynności korygujące (kompensacje). (*Koordynator może być komponentem samodzielnym, zewnętrznym w stosunku do konsumenta)

sacją (ang. *compensation*), które niekoniecznie będą dokładnym przeciwieństwem działań przeznaczonych do anulowania. Jeśli w wyniku pierwotnych działań usługa przekroczyła pewien próg, może ona nie chcieć anulowania wykonanych czynności. Anulowanie efektu może też okazać się niemożliwe, np. jeśli wycofanie działań wymaga od usługi czegoś, co pierwotnie zainicjowało podjęcie tych działań (można to nazwać opłatą za anulowanie), lub upłynęło już zbyt dużo czasu. Przykładowo, jeśli wynikiem działania sagi byłoby wystrzelenie pocisku, kompensację stanowiłoby przerwanie misji i zdetonowanie pocisku w powietrzu — nie można przecież zawrócić pocisku i umieścić go z powrotem w wyrzutni.

Wzorzec Saga jest też czasem określany mianem „Transakcji Długotrwałej". To prawda, że koncepcyjnie możesz traktować sagę jako pojedynczą logiczną jednostkę pracy i że korzysta ona z semantyki transakcyjnej. Jednak saga w rzeczywistości nie przestrzega dogmatów transakcyjnych, takich jak niepodzielność czy izolacja, głównie dlatego, że interakcja jest rozproszona zarówno w czasie, jak i przestrzeni. Przykładowo, kiedy wywołujesz kompensację, może być już za późno na anulowanie pierwotnej czynności, mogą więc wystąpić konsekwencje, takie jak opłaty za anulowanie albo częściowe dostarczenie. Termin „Saga" dobrze odzwierciedla fakt, że interakcja jest długotrwała, a komunikaty są powiązanie.

Spójrzmy, jak mógłby wyglądać scenariusz zamawiania z rysunku 5.12 przy zastosowaniu wzorca interakcji Saga. Na rysunku 5.14 przedstawiono scenariusz, w którym u dostawcy wystąpił brak dostępności zamówionych towarów. W takim przypadku zarówno zamawianie, jak i rozliczanie musi zostać anulowane. Musisz również powiadomić fasadę, że wystąpił problem, i poinformować dostawcę, że zamknąłeś interakcję.



Rysunek 5.14. Scenariusz e-commerce z rysunku 5.12 przemodelowany pod kątem wzorca Saga. Interakcja z usługą „Rozliczenia" i systemem „Dostawca" jest teraz koordynowana przez sagę. Usługa „Zamawianie" może teraz radzić sobie z problemami w bardziej wydajny sposób, anulując zamówienie i powiadamiając o tym fasadę, zamiast liczyć na to, że wszystko będzie dobrze

W tej wersji scenariusza zamawiania ze wzorcem Saga wszystkie zaangażowane usługi („Zamawianie", „Rozliczenia" oraz system „Dostawca") wysyłają powiadomienia, czy są zdolne do wypełnienia sagi, czy nie. Przykładowo, system „Dostawca" wysyła komunikat błędu, aby powiadomić usługę „Zamawianie", że miał problem z przetworzeniem żądania „złożenie zamówienia". Kiedy komponent koordynatora w usłudze „Zamawianie" odbiera komunikat o błędzie, wysyła żądania kompensacji do pozostałych stron (do usług „Zamawianie" i „Rozliczenia"), po czym powiadamia system „Dostawca", że interakcja zakończyła się obsłużeniem błędu.

Fasada jest powiadamiana o niepowodzeniu podczas kompensacji usługi „Zamawianie". Nie jest to zadaniem koordynatora.

W interakcji przedstawionej na rysunku 5.14 konsument usług i usługi kontrolują interakcję wewnętrznie. Dobrym na to sposobem jest zastosowanie wzorca Przepływ Pracy (omówionego w rozdziale 2.), tak aby każda usługa posiadała wewnętrzny przepływ pracy podążający za sekwencją i za różnymi ścieżkami interakcji. Innym wzorcem związanym z Sagą jest wzorzec Rezerwacja (patrz rozdział 6.).

Kolejnym podejściem przy implementacji wzorca Saga jest zastosowanie zewnętrznego koordynatora dla konwersacji — więcej szczegółów znajdziesz w kontekście omawiania wzorca Orkiestracja w rozdziale 7. Różnica semantyczna pomiędzy wewnętrznie koordynowaną implementacją Sagi a implementacją koordynowaną zewnętrznie jest taka, że w tej drugiej koordynator posiada „szeroki obraz" tego, co saga próbuje osiągnąć, podczas gdy w koordynacji wewnętrznej możesz osiągnąć koordynację bez konieczności posiadania pełnego obrazu przez którąkolwiek usługę. Koordynacja wewnętrzna jest bardziej elastyczna, ale trudniejsza w zarządzaniu.

Głównym wysiłkiem przy implementacji wzorca Saga jest podjęcie decyzji odnośnie działań biznesowych i kompensacji. Aby określić, jakie mogą być te działania, możesz zastosować techniki takie jak modelowanie procesów biznesowych. Notacja i modelowanie procesów biznesowych (ang. *Business Process Modeling and Notation* — BPMN) została omówiona w jednym z punktów dotyczących wzorca Orkiestracja w rozdziale 7.

Mimo że główny wysiłek w implementacji wzorca Saga dotyczy strony biznesowej, modelowania procesów biznesowych oraz działań, które będą obsługiwać długotrwałe konwersacje, to jest też kilka technologicznych aspektów związanych z komunikatami i protokołami — przyjrzyjmy się im teraz.

MAPOWANIE TECHNOLOGII

Minimalne wymagania dla wzorca Saga to dodanie komunikatów kompensacyjnych do wszystkich informujących o stanie komunikatów uczestniczących w sadze. Ponownie należy podkreślić, że kompensacja może nie być w stanie anulować pierwotnych działań, ale musi próbować minimalizować skutek tych działań.

Wewnętrzne przetwarzanie komunikatów kompensacyjnych różni się w zależności od tego, co musi zostać zrobione w celu usunięcia efektu pierwotnego komunikatu. Zazwyczaj lepszym wyjściem jest ustawianie statusów „anulowane" niż usuwanie rekordów, szczególnie na poziomie bazy danych, ponieważ pierwotne działanie mogło uruchomić inne procesy biznesowe i działania bazujące na tych rekordach. Przykładowo, jeśli rezultatem komunikatu jest dodanie zamówienia, inna usługa mogła już wygenerować rachunek. Są szanse, że wygenerowanie rachunku miało miejsce w ramach tej samej sagi, ale usługa „Zamawianie" może o tym nie wiedzieć lub tego nie kontrolować. Dokonywanie zmian, które pozostawiają po sobie ślad (np. ustawienie statusu „anulowano"), jest lepszym wyjściem niż usuwanie rekordów, ponieważ umożliwia to ręczne rozwiązywanie problemów, gdy zachodzi taka potrzeba. Zwróć uwagę, że w niektórych branżach, takich jak bankowość, prawo obliguje raczej do rejestrowania procesów anulowania jako zmian, a nie do usuwania lub

poprawiania oryginalnych rekordów. (Więcej informacji na temat tego, że nie należy usuwać rekordów, znajdziesz w artykule „Accountants Don't Use Erasers" autorstwa Pata Hellanda wyszczególnionym w podrozdziale „Dalsza lektura").

Kolejnym typem komunikatów istotnym dla wzorca Saga są komunikaty o niepowodzeniu. Kiedy masz do czynienia z prostą interakcją punkt-punkt pomiędzy usługami, odpowiedź lub reakcja wysyłane przez wywołaną usługę wystarczają do przekazania istoty problemu. Wywołujący usługę konsument usług, który rozumie kontrakt usługi, może zorientować się, że coś jest nie tak, i podjąć odpowiednie działania. Jednak jeśli implementujesz wzorzec Saga, możesz mieć więcej niż dwie zaangażowane strony oraz koordynatora. Koordynator nie ma tak dużej świadomości biznesowej, jak logika biznesowa usługi, ale definiuje komunikaty kontrolujące w celu zrozumienia statusu interakcji.

Jak pewnie wiesz (lub zdążyłeś się już zorientować), usługi sieciowe są postrzegane jako podstawowa technologia do implementacji SOA, w czym wzorzec Saga również zbytnio się nie różni. Stos protokołów WS-* wprowadził protokół WS-BusinessActivity jako element WS-Coordination.

Protokół WS-BusinessActivity posiada dwa warianty:

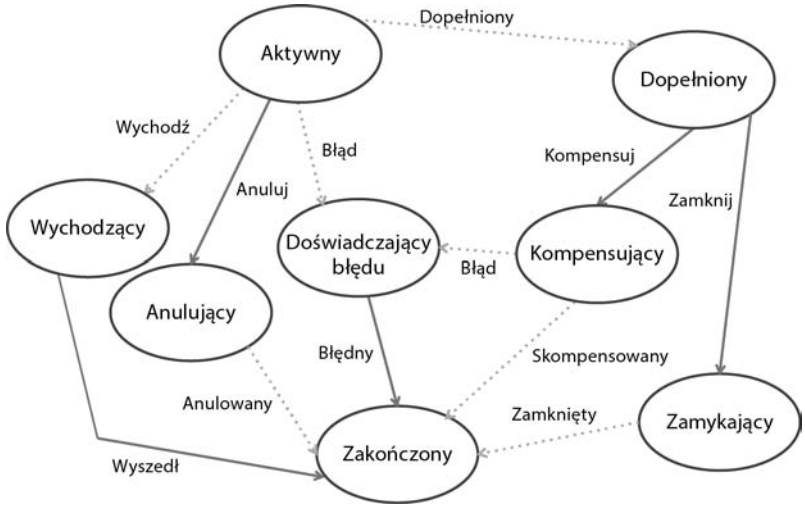
- **Business Agreement with Coordinator Completion** (uzgodnienia biznesowe dopełniane przez koordynatora) — Koordynator decyduje i powiadamia uczestników, kiedy dopełnia się ich rola w ramach danego działania. Takie podejście jest nieco bardziej uporządkowane.
- **Business Agreement with Participant Completion** (uzgodnienia biznesowe dopełniane przez uczestników) — Uczestnicy decydują, kiedy dopełnia się ich rola w ramach danego działania. To podejście charakteryzuje się luźniejszymi powiązaniami oraz kosztem, którym są zwiększone szanse na kompensację.

WS-BusinessActivity definiuje uporządkowany protokół oraz stany zarówno dla uczestniczących usług, jak i koordynatora. Definiuje również dwa rodzaje koordynatorów:

- `AtomicOutcome` — Wszyscy uczestnicy muszą zamknąć (zatwierdzić) interakcję lub przeprowadzić kompensację.
- `MixedOutcome` — Koordynator zajmuje się każdym uczestnikiem osobno.

Na rysunku 5.15 przedstawiono przejścia stanów dla uczestniczących usług przy zastosowaniu protokołu WS-BusinessActivity w wariancie dopełnianym przez uczestników.

Inną istotną opcją technologiczną służącą do implementacji wzorca Saga jest zastosowanie języka BPEL (ang. *Business Process Execution Language*) lub jego implementacja WS-* znana jako WS-BPEL (lub BPEL4WS w poprzednich wersjach). Ponadto możesz również zastosować silnik orkiestracji niekompatybilny z BPEL. Tego typu mapowanie technologii dotyczy wspomnianej uprzednio koordynacji zewnętrznej i jest opisane bardziej szczegółowo w kontekście wzorca Orkiestracja w rozdziale 7.



Rysunek 5.15. Schemat stanów z punktu widzenia usługi uczestniczącej przy wykorzystaniu protokołu WS-BusinessActivity w wariancie dopełnianym przez uczestników. Przejścia stanów mogą być wynikiem decyzji podejmowanych przez usługę (linie wykropkowane) lub przez komunikaty od koordynatora (linie ciągłe)

ATRYBUTY JAKOŚCIOWE

Głównym powodem do zastosowania wzorca Saga jest potrzeba zwiększenia integralności systemu. Jak wspomniano w poprzednich punktach, transakcje są problematyczne, kiedy w ogóle mamy do czynienia ze środowiskami rozproszonymi, a problem ten jest jeszcze większy w przypadku SOA. Niemniej jednak nadal chcemy mieć możliwość koordynowania usług i posiadać znaczące interakcje. Koordynując to zachowanie oraz obsługę błędów, możesz wprowadzić niezawodne, przewidywalne i długotrwałe konwersacje.

W środowisku rozproszonym stosunkowo ciężko jest przewidzieć, jaki będzie wynik złożonej interakcji. Dotyczy to szczególnie innych wzorców, takich jak Odwrócenie Komunikacji (omówiony w podrozdziale 5.3). Wzorzec Saga wprowadza do interakcji pewną kontrolę i pilnuje, aby wynik złożonej interakcji odpowiadał znanym schematom (dopełniony lub skompensowany).

Rezultatem zwiększonej przewidywalności jest również wzrost poprawności. Kiedy możesz przewidzieć zachowanie systemu, łatwiej jest przygotować testy systemu, aby zweryfikować, czy otrzymasz pożądany wynik.

Tabela 5.5 prezentuje przykładowe scenariusze dla wymienionych atrybutów jakościowych.

Napisanie logiki kompensacji jest dość skomplikowane. Wraz z upływem czasu liczba zmian w usłudze może stać się dość duża, co utrudnia osiągnięcie przewidywalności, kiedy próbujesz anulować wczesne zmiany. Jednym ze sposobów radzenia sobie z tą kwestią jest zaimplementowanie wzorca Rezerwacja, o którym przeczytasz w następnym rozdziale.

Tabela 5.5. Atrybuty jakościowe wzorca Saga i przykładowe scenariusze

Atrybut jakościowy	Konkretny atrybut	Przykładowy scenariusz
Integralność	Poprawność	Bez względu na warunki zamówienie przetwarzane przez system zostanie rozliczone
Integralność	Przewidywalność	W normalnych warunkach szanse wystawienia konsumentowi rachunku za anulowane zamówienie wynoszą mniej niż 5%
Niezawodność	Obsługa błędów	Po wznowieniu działania po przerwaniu komunikacji wszystkie procesy, które zostały zakłócone, pozostaną spójne

5.5. Podsumowanie

Jedną z cech wyróżniających wszystkie wzorce przedstawione w tym rozdziale jest to, że nie są one nowe. Wszystkie wzorce interakcji wyprzedzają SOA o wiele lat. Mimo to poświęciłem kilkadziesiąt stron na ich prezentację, zamiast po prostu odwołać się do znakomitej książki Hohpe'a i Woolfa *Enterprise Integration Patterns*, która również je omawia. Powodem tego jest fakt, że chociaż są to stosunkowo proste i dobrze znane wzorce, to każdy z nich ma pewne aspekty, które nieco go komplikują przy próbie implementacji w SOA i dostosowaniu do zasad tej architektury.

- **Żądanie/Odpowiedź** — Ten wzorzec dotyczy komunikacji synchronicznej, ale w SOA lepiej jest stosować interakcje oparte na dokumentach. Różni się to znacznie od interakcji opartych na wywołaniach RPC, które są standardem komunikacji synchronicznej w tradycyjnych architekturach rozproszonych.
- **Żądanie/Reakcja** — Wzorzec ten implementuje komunikację asynchroniczną. Ponownie jest to prosty wzorzec, ale jego implementacja może być skomplikowana, jeśli masz do czynienia z konsumentami nieobsługującymi wywołań zwrotnych.
- **Odwrócenie Komunikacji** — Ten wzorzec implementuje zdarzenia, ale może przysporzyć kilka problemów, takich jak implementacja na bazie protokołów transportowych nieobsługujących zdarzeń. Innym interesującym aspektem jest dostarczanie strumienia zdarzeń.
- **Saga** — Saga jest sposobem, aby usługi mogły osiągnąć rozproszony konsensus bez polegania na transakcjach rozproszonych.

Dwa kolejne rozdziały poświęcone są mniej podstawowym wzorcom interakcji. Niektóre z nich są komplementarne do omówionych tutaj wzorców. Jest to np. wzorzec Rezerwacja z rozdziału 6., który uzupełnia wzorzec Saga, czy też wzorzec Zagregowane Raportowanie z rozdziału 7., który korzysta z wzorca Odwrócenie Komunikacji. Pozostałe wzorce, którymi się zajmiemy, takie jak wzorzec Fasada Kompozytowa z rozdziału 6., dotyczą aspektów interakcji i agregacji wykraczających poza bazowe wzorce wymiany komunikatów.

5.6. Dalsza lektura

Gregor Hohpe i Bobby Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions* (Addison-Wesley Professional, 2003).
Ta książka omawia w ogólnym kontekście podstawowe wzorce integracyjne, z których wiele ma również zastosowanie w SOA.

Interfejsy API Google Data, *http://code.google.com/apis/gdata/overview.html*.
Protokół GData (ang. *Google Data Protocol*) jest przykładem dokumentowo-centrycznego protokołu interakcji z usługami.

WZORZEC ODWRÓCENIE KOMUNIKACJI

Arnon Rotem-Gal-Oz, „Bridging the Gap Between BI & SOA", *http://www.infoq.com/articles/BI-and-SOA*.
Ten artykuł pokazuje zastosowanie wzorca Odwrócenie Komunikacji (a także wzorca Zagregowane Raportowanie).

Matt Welsh, „SEDA: An Architecture for Highly Concurrent Server Applications", *http://www.eecs.harvard.edu/~mdw/proj/seda/*.
Połączenie wzorców Odwrócenie Komunikacji oraz Potoki Równoległe jest implementacją SEDA dla SOA.

WZORZEC SAGA

Pat Helland, „Accountants Don't Use Erasers", *Pat Helland's WebLog*, *http://blogs.msdn.com/b/pathelland/archive/2007/06/14/accountants-don-t-use-erasers.aspx*.
Pat Helland objaśnia zalety zachowywania poprzednich stanów.

Skorowidz

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Z

Ż

Obowiązkowa lektura każdego projektanta!

SOA (ang. Service Oriented Architecture) to gorący skrót ostatnich lat. Koncepcja oferowania niezależnych usług do określonych zadań zdobyła sobie ogromną popularność. Takie podejście pozwala na tworzenie elastycznych systemów informatycznych, które są znacznie łatwiejsze w utrzymaniu, zaprojektowaniu i wykonaniu od tradycyjnych rozwiązań. Ponadto udostępnienie pojedynczych serwisów innym projektantom może przynieść dodatkowe dochody lub zwiększyć atrakcyjność Twojej aplikacji.

Prawda, że brzmi zachęcająco? Po przeczytaniu tej książki nie oprzesz się wrażeniu, że jest to jedyna słuszna droga w zakresie wytwarzania oprogramowania. W trakcie lektury dowiesz się, jak zapewnić najwyższą jakość, dostępność i przepustowość tworzonych usług. Poznasz kolejne wzorce, które pozwolą Ci zaprojektować przejrzysty i bezpieczny system. Integracja usług, wymiana danych między serwisami, tworzenie klienta usług to tylko niektóre z poruszanych zagadnień. Osobny rozdział został poświęcony antywzorcom — to obowiązkowy punkt lektury, bo przecież musisz wiedzieć, jak tego nie robić! Sprawdź tę kapitalną pozycja dla każdego projektanta i programisty chcącego tworzyć nowoczesne systemy informatyczne.

Dowiedz się:

- jak stworzyć niezawodną i wydajną usługę
- co osiągniesz dzięki stosowaniu podejścia SOA
- jak wymieniać dane pomiędzy różnymi usługami
- jak złożyć wszystkie usługi w jeden system

MANNING

Nr katalogowy: 14581

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

helion.pl
księgarnia internetowa

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena 54,90 zł

ISBN 978-83-246-7050-5

9 788324 670505

Informatyka w najlepszym wydaniu