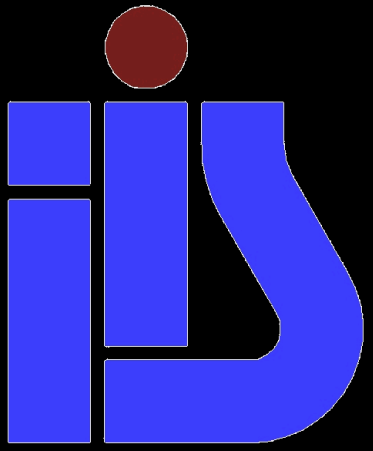


Od Inżynierii Komponentowej do Usług Internetowych

Inżynieria Komponentowa

Component-based software engineering

Inżynieria komponentowa

- Podejście do wytwarzania oprogramowania polegająca na powtórnym użyciu elementów zwanych komponentami oprogramowania.
- Pojawiła się jako odpowiedź na porażkę obiektowości w kontekście powtórnego użycia w większej skali (Pojedyncze klasy obiektów są zbyt szczegółowe i specyficzne).
- Komponenty są bardzie abstrakcyjne niż klasy obiektów i mogą być dostawcami złożonych usług (jako niezależne jednostki).

Podstawy Inżynierii Komponentowej

- Niezależne komponenty definiowane przez interfejsy.
- Standardy komponentów w celu umożliwienia ich integracji.
- Warstwa pośrednia wspierająca interoperacyjność komponentów.
- Proces rozwoju, który jest nakierowany na powtórne użycie.

Zasady projektowanie a Inżynieria Komponentowa

- Poza zasadą wielokrotnego użycia Inżynieria Komponentowa „wspiera”:
  - Niezależność – Komponenty są niezależne – nie kolidują ze sobą;
  - Hermetyzacja –
    - Implementacja komponentów jest ukryta;
    - Komponenty komunikują się za pośrednictwem interfejsów;
  - Współdzielenie
    - Platformy komponentowe są współdzielone przez komponenty co pozwala na redukcję kosztów rozwoju.

Standardy komponentów

- Abu komponenty mogły się ze sobą komunikować oraz współpracować potrzebne są standardy.
- Niestety nie istnieje jeden standard:
  - Enterprise Java Beans (JEE)
  - Microsoft's COM i .NET
  - CORBA's CCM
- W praktyce utrudnia to asymilację podejścia.
  - Nie ma możliwości współpracy komponentów wykorzystujących różne podejścia.

Problemy Inżynierii Komponentowej

- Wiarygodność
- Certyfikacja komponentów
- Przewidywanie własności komponentów
- Kompromisy wymagań

Komponenty

- Komponenty udostępniają usługę niezależnie od miejsca uruchamiania i wykorzystanego języka programowania
  - Komponent to niezależna, wykonywalna jednostka, która może składać się z jednego lub wielu wykonywalnych obiektów;
  - Interfejs komponentu jest publiczny i wszelka interakcja odbywa się za jego pośrednictwem;

Definicje komponentu

- Councill and Heinmann:
  - *A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.*
- Szyperski:
  - *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.*

Charakterystyka komponentów (1)

Cecha	Opis
Standaryzacja	Komponent wykorzystywane w procesie inżynierii komponentowej musi być zgodny ze standardem modelu komponentowego. Model może definiować interfejsy komponentu, metadane komponentu, sposób dokumentowania, kompozycji oraz rozmieszczania komponentu.
Niezależność	Komponent powinien być niezależny – powinno być możliwe komponowanie oraz rozmieszczanie komponentów bez potrzeby wykorzystania innych komponentów. W sytuacji gdy komponent wymaga zewnętrznych usług powinny być one wyszczególnione w ramach definicji interfejsu wymaganego.
Kompozycyjność	Aby komponent był kompozycyjny, wszystkie zewnętrzne interakcji musza być wykonywane za pośrednictwem publicznych interfejsów. Dodatkowo komponent musi dostarczać informacji o samym sobie (np. dostępne metody i atrybuty).

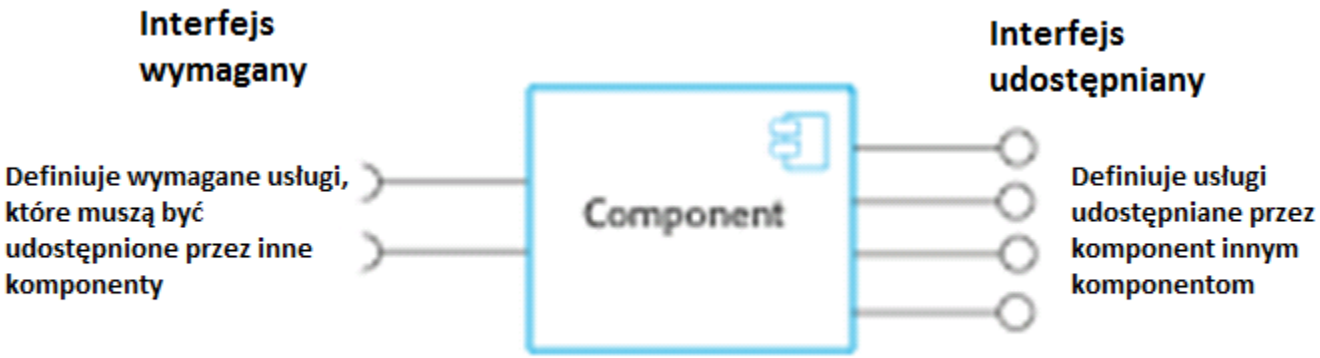
Charakterystyka komponentów (2)

Cecha	Opis
Swoboda rozmieszczania/ wdrażania	Aby można było komponent rozmieścić (umieścić w kontenerze), musi być samowystarczalny. Możliwe jest wówczas operowanie komponentem jako niezależną jednostką w ramach platformy implementującej model komponentowy. Oznacza to zazwyczaj, że komponent ma postać binarną i nie musi być kompilowany przed rozmieszczeniem. Jeżeli komponent jest implementowany jako usługa nie musi być rozmieszczany przez użytkownika – zadanie to powinno leżeć po stronie dostawcy usługi.
Udokumentowanie	Komponent powinien być udokumentowany w taki sposób aby jego potencjalni użytkownicy mogli, na podstawie dokumentacji, ocenić czy spełnia on ich potrzeby. Dokumentacja powinna definiować składnię i semantykę interfejsów komponentu.

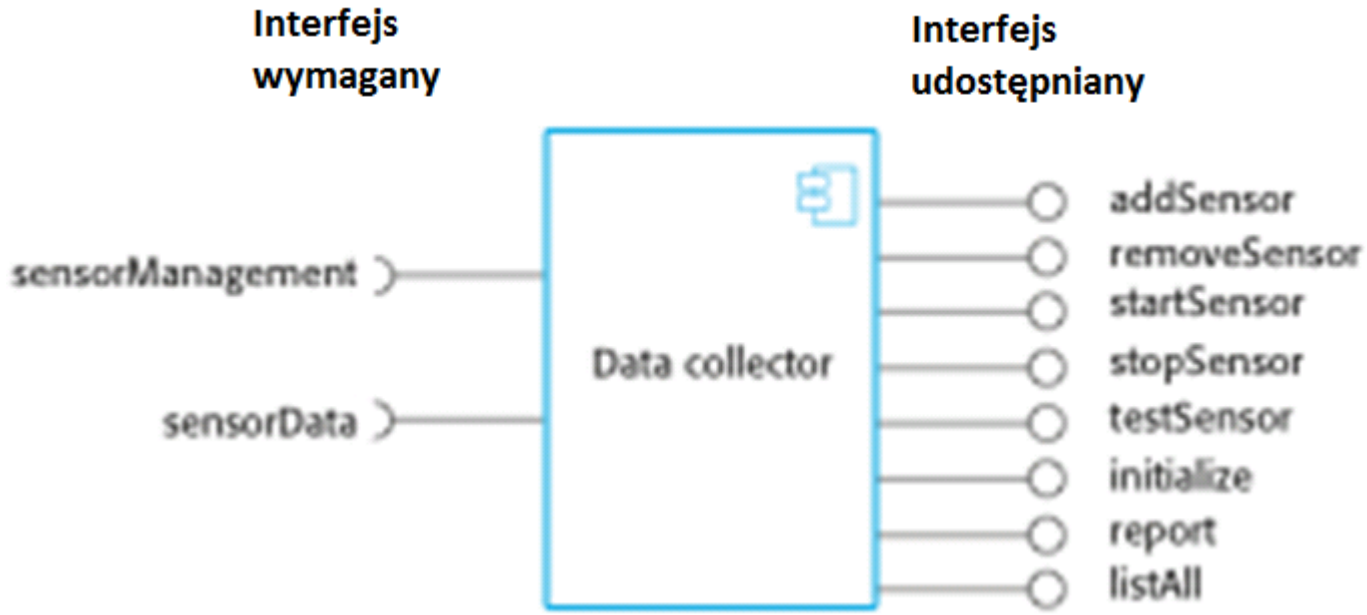
Komponent jako dostawca usług

- Komponent jest niezależną, wykonywalną jednostką. Nie musi być kompilowany przed użyciem.
- Usługa oferowana przez komponent jest dostępna za pośrednictwem interfejsu i cała interakcja z komponentem odbywa się za jego pośrednictwem.
- Interfejs komponentu wyrażony jest w terminach parametryzowanych operacji. Wewnętrzny stan komponentu nie jest eksponowany.

Interfejsy komponentu w UML



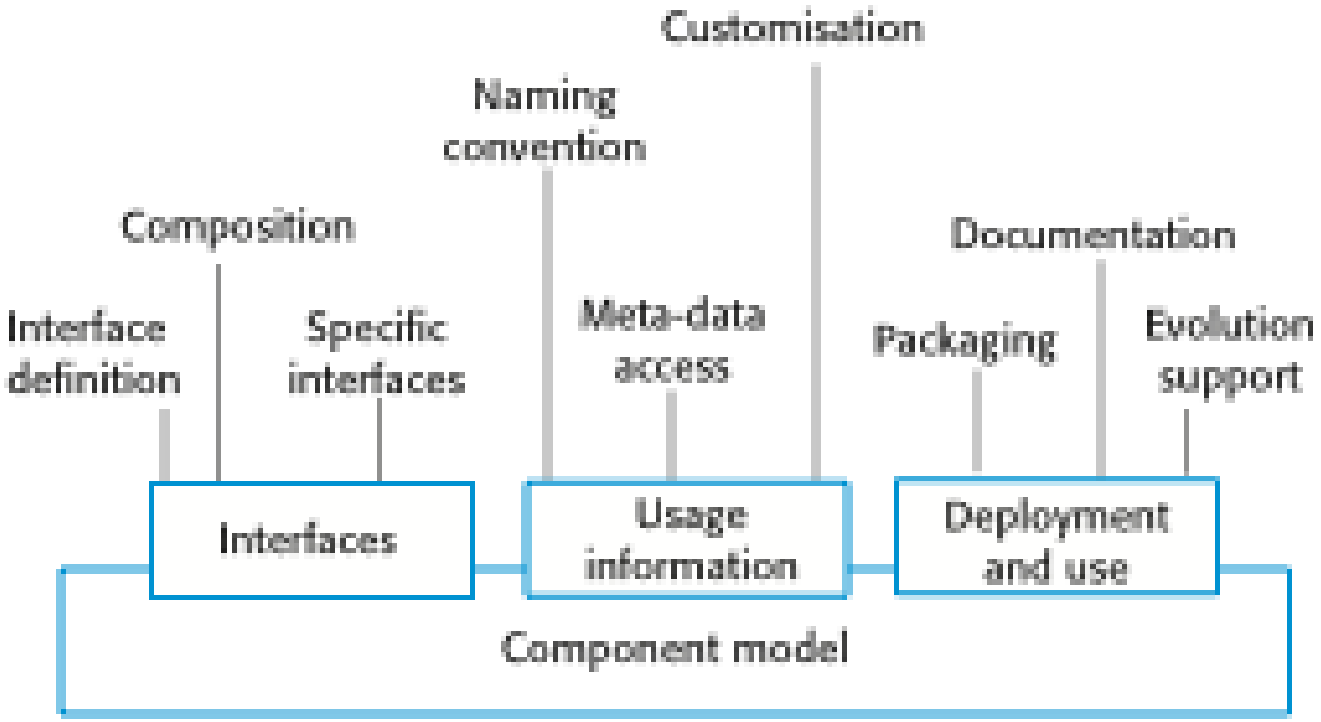
Model UML komponentu „Data collector”



Modele komponentowe

- Model komponentowy to definicja standardów implementacji, dokumentacji i rozmieszczania komponentów.
- Przykładowe modele komponentowe
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - CORBA Component Model
- Model definiuje w jako sposób powinny być definiowane interfejsy oraz specyfikuje elementy, które powinny pojawić się w jego ramach.

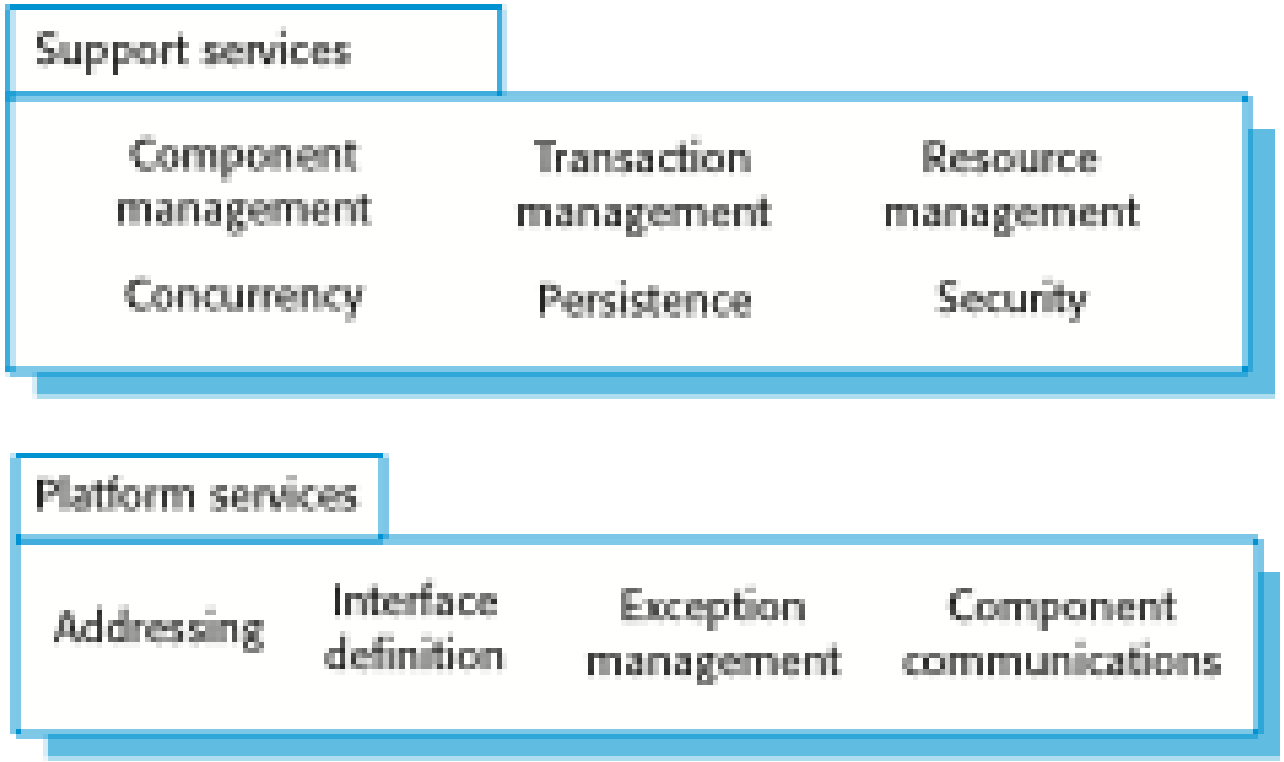
Podstawowe elementy modelu komponentowego



Wsparcie warstwy pośredniej

- Modele komponentowe stanowią podstawę warstwy pośredniej, które udostępnia mechanizmy wspierające uruchamianie komponentów.
- Implementacja modelu komponentowego udostępnia:
  - Usługi platformy, które pozwalają na komunikację komponentów zdefiniowanych zgodnie z modelem.
  - Inne usługi, które są niezależne od usług konkretnych aplikacji i mogą być wykorzystywane przez różne komponenty.
- Aby możliwe był wykorzystanie usług udostępnianych przez implementację modelu, komponenty muszą być rozmieszczane w kontenerach.
  - Kontener udostępnia zestaw (standardowych) interfejsów pozwalających na dostęp do implementacji usług platformy.

Usługi warstwy pośredniej definiowane w ramach modelu komponentowego

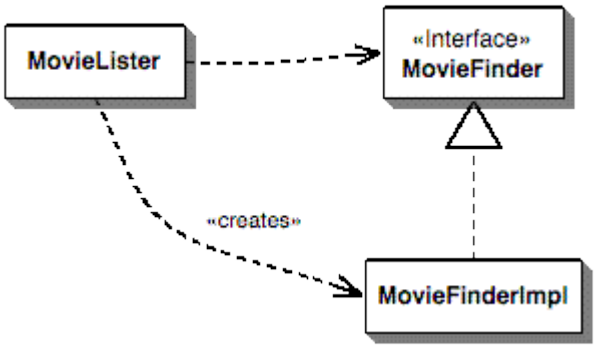


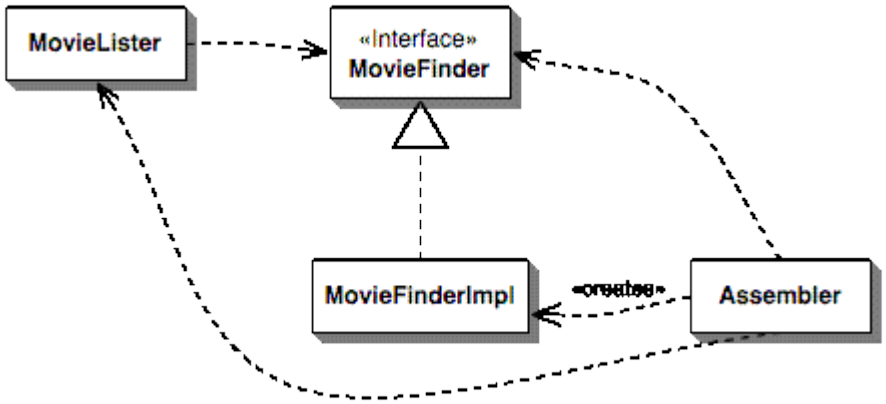
Proces Inżynierii Komponentowej

- Procesy wspierające inżynierię oprogramowania bazującą na komponentach.
  - Rozwój dla wielokrotnego użycia
  - Rozwój z wielokrotnym użyciem

„Lekkie" modele

- Kontenery „odwróconego sterowania"
- POJO
- Spring Framework, Google Guice, Unity

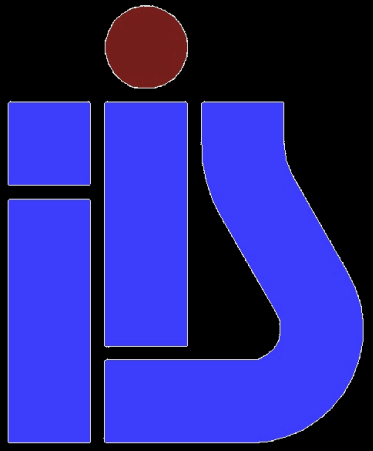




http://www.martinfowler.com/articles/injection.html

Podsumowanie

- Inżynieria komponentowa to, bazujące na powtórnym użyciu, podejście do definiowania i implementacji systemu z luźno powiązanych komponentów.
- Komponent to jednostka oprogramowania, której funkcjonalność oraz zależności zewnętrzne są w całości definiowane przez jej interfejs.
- Model komponentowy definiuje zestaw standardów, które muszą być przestrzegane przez dostawców komponentów oraz uwzględniane przez komponujących systemy.
- Podstawowe procesy w ramach inżynierii komponentowej to inżynieria dla wielokrotnego użycia oraz inżynieria z wielokrotnym użyciem.



Architektura zorientowana usługowo

Zagadnienia

- Usługi jako komponenty wielokrotnego użycia
- Inżynieria usług
- Rozwój oprogramowania z wykorzystaniem usług

Usługi Internetowe (ang. Web services)

- Usługa Internetowa to szczególne wystąpienie usługi:

  Usługa:

  "działanie podejmowane w celu zaspokojenia określonej potrzeby. Usługa może obejmować czynności niematerialne, jak porada techniczna czy wydajność oraz materialne, jak wykonawstwo konkretnych przedmiotów. Usługa zazwyczaj nie nakłada wymagania posiadania zasobów produkcyjnych przez użytkowników usługi."

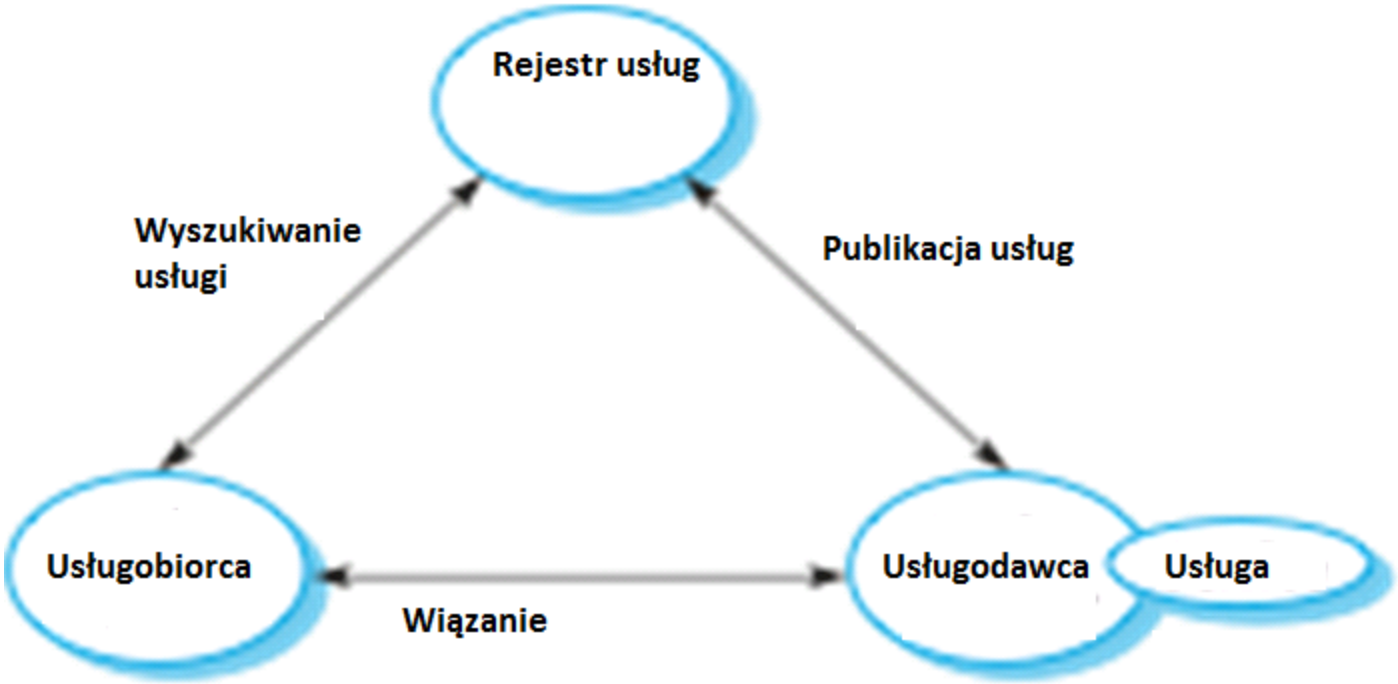
- Istotą usługi jest jej **niezależność od aplikacji ją wykorzystującej**.

- Dostawcy usług mogą rozwijać specjalizowane usługi oferowane zewnętrznym organizacjom.

Architektura zorientowana usługowo (Service Oriented Architecture SOA)

- Sposób rozwijania systemów rozproszonych, gdzie komponenty są niezależnymi usługami.
- Usługi mogą być wykonywane na różnych maszynach przez różnych dostawców.
- Standardowe protokoły pozwalają na komunikację z usługą oraz wymianę informacji.

Architektura zorientowana usługowo



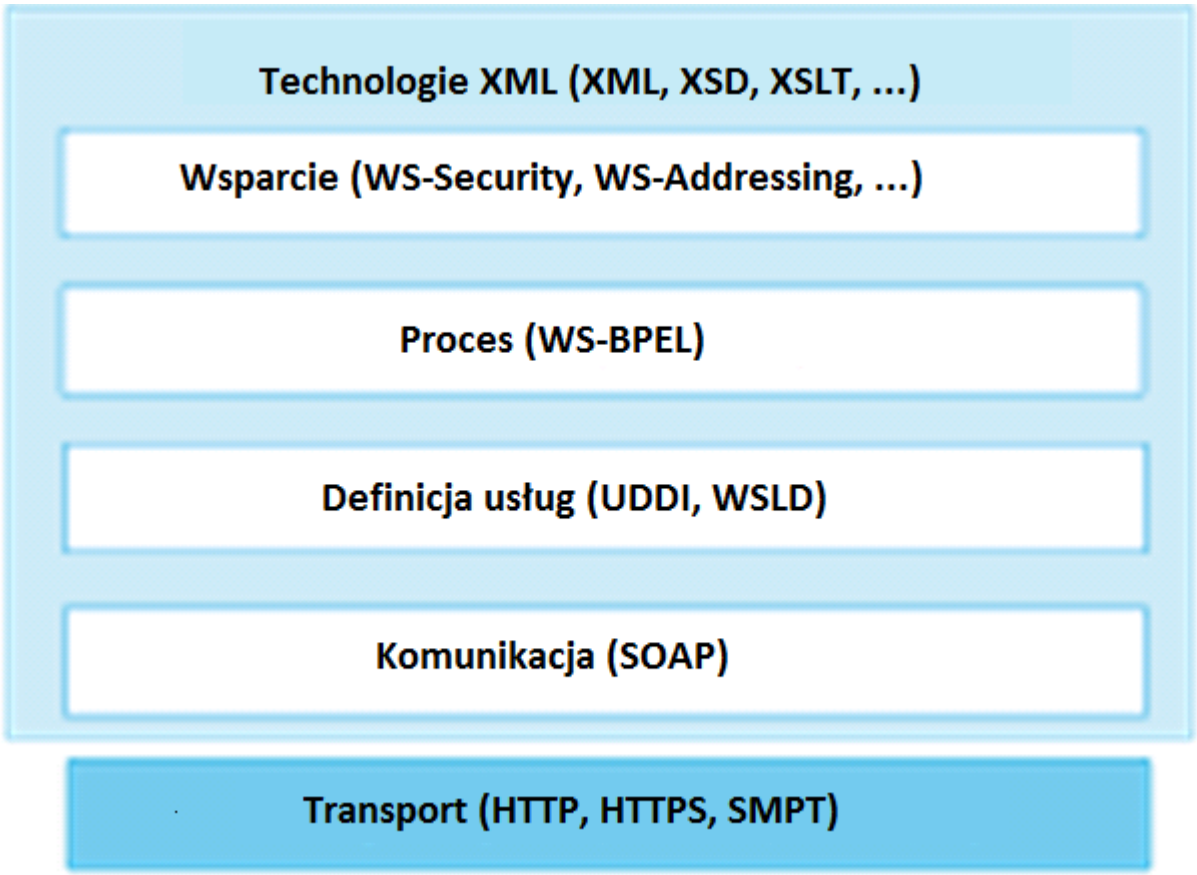
Zalety SOA

- Usługi mogą być dostarczane lokalnie lub zewnętrznie
- Usługa jest niezależna od języka
- Systemy spadkowe (ang. legacy) mogą zostać stosunkowo łatwo wykorzystane
- Uproszczenie mechanizmów wymiany informacji pozwala na wykonywanie obliczeń przekraczających granice organizacji.

Standardy Usług Internetowych (1)

- SOAP (ang. Simple Object Access Protocol)
  - Standard komunikacji z usługami bazujący na abstrakcji komunikatów.
- WSDL (ang. Web Service Definition Language)
  - Standard pozwalający na opisanie interfejsu usługi oraz sposobu wiązania
- WS-BPEL (ang. Web Services – Business Process Execution Language)
  - Standard języków pozwalających na definiowanie kompozycji usług internetowych (workflow)

Standardy Usług Internetowych (2)



Krytyka standardów Usług Internetowych

- Duża liczba technologii
- Zbyt ogólne
- Problemy wydajnościowe
- Proste aplikacje wymagają dodatkowych i często niepotrzebnych działań

Usługi Internetowe w stylu REST

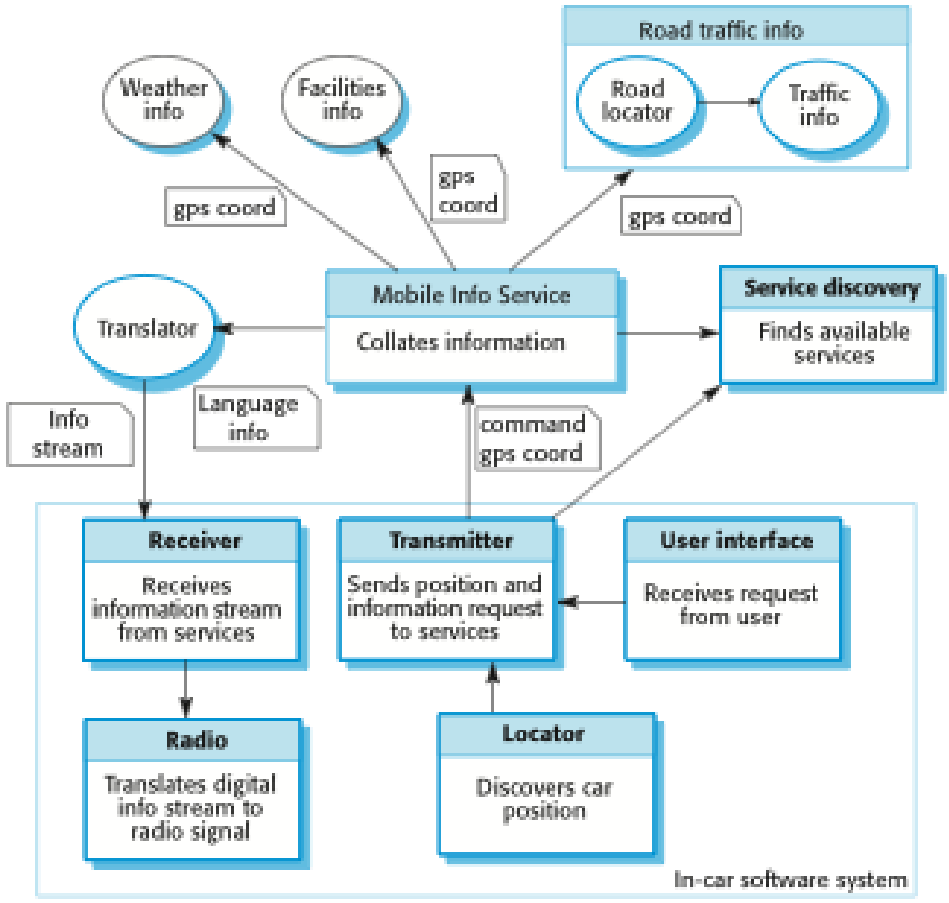
- REST (REpresentational State Transfer) – wzorzec architektoniczny bazujący na przesyłaniu przez serwer reprezentacji zasobu.
- Wzorzec opiera się na istniejącej infrastrukturze Internetowej (nie wprowadza nowych protokołów) co upraszcza implementację usług sieciowych.
- Usługi Internetowe typu REST (określane jako RESTful) są mniej wymagające implementacyjnie w porównaniu do tzw. dużych Usług Internetowych. Są wykorzystywane przez organizacje implementujące systemy bazujące na usługach, ale nie wykorzystujące usług dostępnych zewnętrznie.

Przykładowy scenariusz usługi

- System samochodowy dostarcza kierowcy informacji o warunkach pogodowych, ruch ulicznym, lokalne informacje, itd. System jest połączony z odbiornikiem radiowym w taki sposób, że informacje są dostarczane w postaci sygnału stacji radiowej.

- Samochód jest wyposażony w odbiornik GPS pozwalający na określanie pozycji. Na podstawie tej informacji system wybiera odpowiednie do serwisy informacyjne. Informacja może być dostarczana w wybranym języku.

Samochodowy system informacyjny oparty na usługach



Zalety SOA dla przykładowej aplikacji

- Decyzja o wyborze dostawcy usług oraz samym zestawie usług nie musi być ostatecznie podejmowana ani w trakcie implementacji ani wdrażania systemu.
  - W trakcie podróży, w celu znalezienia odpowiedniej usługi, oprogramowanie wykorzystuje serwis lokalizacji (ang. discovery service).
  - Wykorzystanie usługi pozwalającej na tłumaczenie system jest w stanie dostarczyć informacje lokalne w stosunku do bieżącego punktu podróży w języku specyficznym dla kierowcy.

Inżynieria oprogramowania zorientowanego usługowo (Service-oriented software engineering)

- Rozwój oprogramowania zorientowanego usługowo wymaga specyficznego podejścia, które kładzie nacisk na:
  - Inżynierię usług czyli rozwój usług dla wielokrotnego użycia (Software development **for** reuse)
  - Rozwój oprogramowania w oparciu o istniejące usługi (Software development **with** reuse)

Usługi jako komponenty powtórnego użycia

- Usługa może by zdefiniowana w następujący sposób:
  - *Luźno połączone, komponenty wielokrotnego użycia, hermetyzujące funkcjonalność, która może być rozpowszechniana i dostępna programowo (API). Usługa Internetowa to usługa, która jest dostępna za pośrednictwem standardów internetowych i protokołów opartych na XML.*
- Usługa a komponent
  - Usługa jest niezależna
  - Usługa nie specyfkuje interfejsów wymaganych do działania
  - Interakcja z usługą opiera się na przekazywaniu komunikatów (zapisanych z wykorzystaniem XML).

Język opisu Usługi Internetowej

- WSDL (ang. Web Service Description Language).
- Specyfikacja WSDL definiuje:
  - „Co” - Jakie operacje udostępnia usługa i jaki jest format komunikatów wysyłanych i odbieranych przez usługę.
  - „Jak” - Jak uzyskać dostęp do usługi – mapowanie abstrakcyjnego interfejsu na konkretny zestaw protokołów.
  - „Gdzie” - Lokalizacja usługi – zazwyczaj w postaci URI (Universal Resource Identifier)

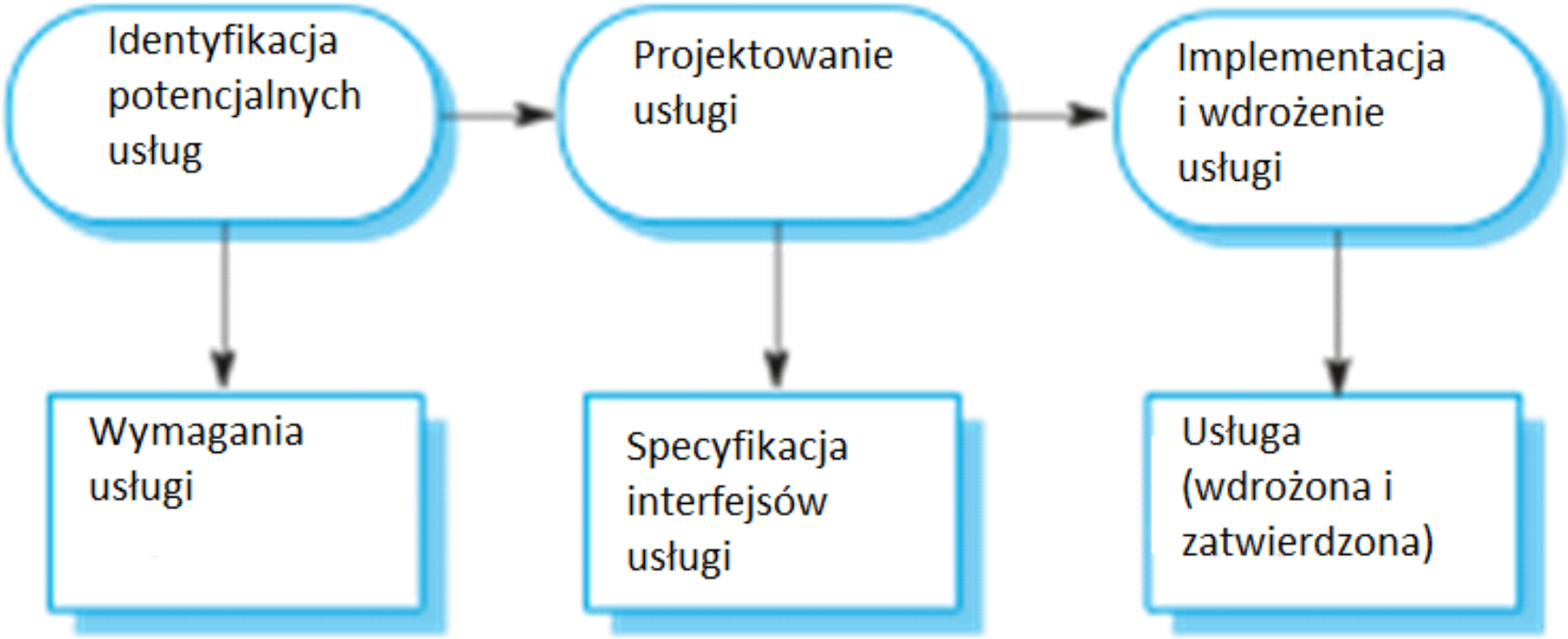
Fragment opisu Usługi Internetowej w języku WSDL

```
<types>
   <xs: schema targetNameSpace = “http://.../weathns”
      xmlns: weathns = “http://…/weathns” >
      <xs:element name = “PlaceAndDate” type = “pdrec” />
      <xs:element name = “MaxMinTemp” type = “mmtrec” />
      <xs: element name = “InDataFault” type = “errmess” />

      <xs: complexType name = “pdrec”
      <xs: sequence>
      <xs:element name = “town” type = “xs:string”/>
      <xs:element name = “country” type = “xs:string”/>
      <xs:element name = “day” type = “xs:date” />
      </xs:complexType>
   </schema>
</types>

<interface name = “weatherInfo” >
   <operation name = “getMaxMinTemps” pattern = “wsdlns: in-out”>
      <input messageLabel = “In” element = “weathns: PlaceAndDate” />
      <output messageLabel = “Out” element = “weathns:MaxMinTemp” />
      <outfault messageLabel = “Out” element = “weathns:InDataFault” />
   </operation>
</interface>
```

Inżynieria usług

- Proces rozwoju usługi w celu jej ponownego wykorzystania w aplikacji zorientowanej usługowo
  - Usługa musi być zaprojektowana w postaci abstrakcji ponownego użycia, która może być wykorzystana w różnych systemach.
  - Wraz abstrakcjami muszą zostać zaprojektowane ogólnie przydatne funkcje. Sama usługa musi być usług musi być odporna na zakłócenia i niezawodna.
  - Wymagane jest udokumentowanie usługi w taki sposób aby można ją było zlokalizować i zrozumieć.

Proces inżynierii usługi



Typy usług

- Usługi ogólne (ang. Utility services)
  - usługi implementujące ogólne funkcjonalności wykorzystywane w wielu procesach biznesowych
- Usługi biznesowe (ang. Business services)
  - usługi powiązane ze specyficzną funkcją biznesową.
- Usługi koordynacyjne (ang. Coordination or process services)
  - Usługi powiązane z ogólnymi procesami biznesowymi uwzględniające różnych aktorów i wiele aktywności.

Usługi zorientowane na zadania i usługi zorientowane na dane

- Usługi zorientowane na zadania (ang. Task-oriented services) to usługi powiązane z jakimś działaniem.
- Usługi zorientowane na dane (ang. Entity-oriented services) to usługi postrzegane jako obiekty. Powiązane są z obiektem biznesowym (np. formularzem aplikacyjnym).
- Usługi ogólne i funkcje biznesowe mogą być zorientowane na zadania lub dane. Usługi koordynacyjne są zawsze zorientowane na zadania.

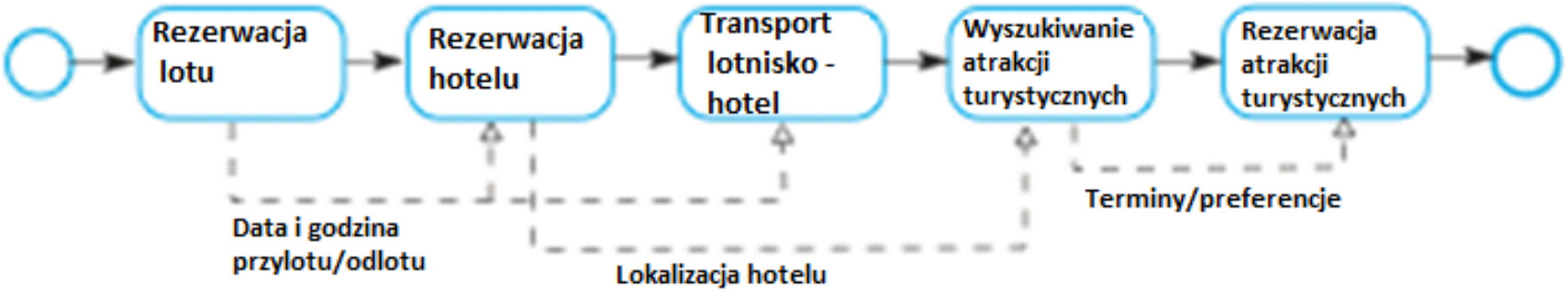
Klasyfikacja usług

	Ogólne (Utility)	Funkcje Biznesowe	Koordynacja procesu
Zadanie (Task)	Konwerter walut Lokalizator pracowników	Zatwierdzanie formularzy zgłoszeniowych Sprawdzanie ratingów	Przetwarzanie kosztów delegacji Realizacja zapłaty dla zewnętrznego dostawcy
Jednostka (Entity)	Kontroler poprawności stylu w dokumencie Konwerter formularza Internetowego do dokumentu XML	Formularze wydatków Formularz zgłoszeniowy studenta	

Rozwój oprogramowania z wykorzystaniem usług

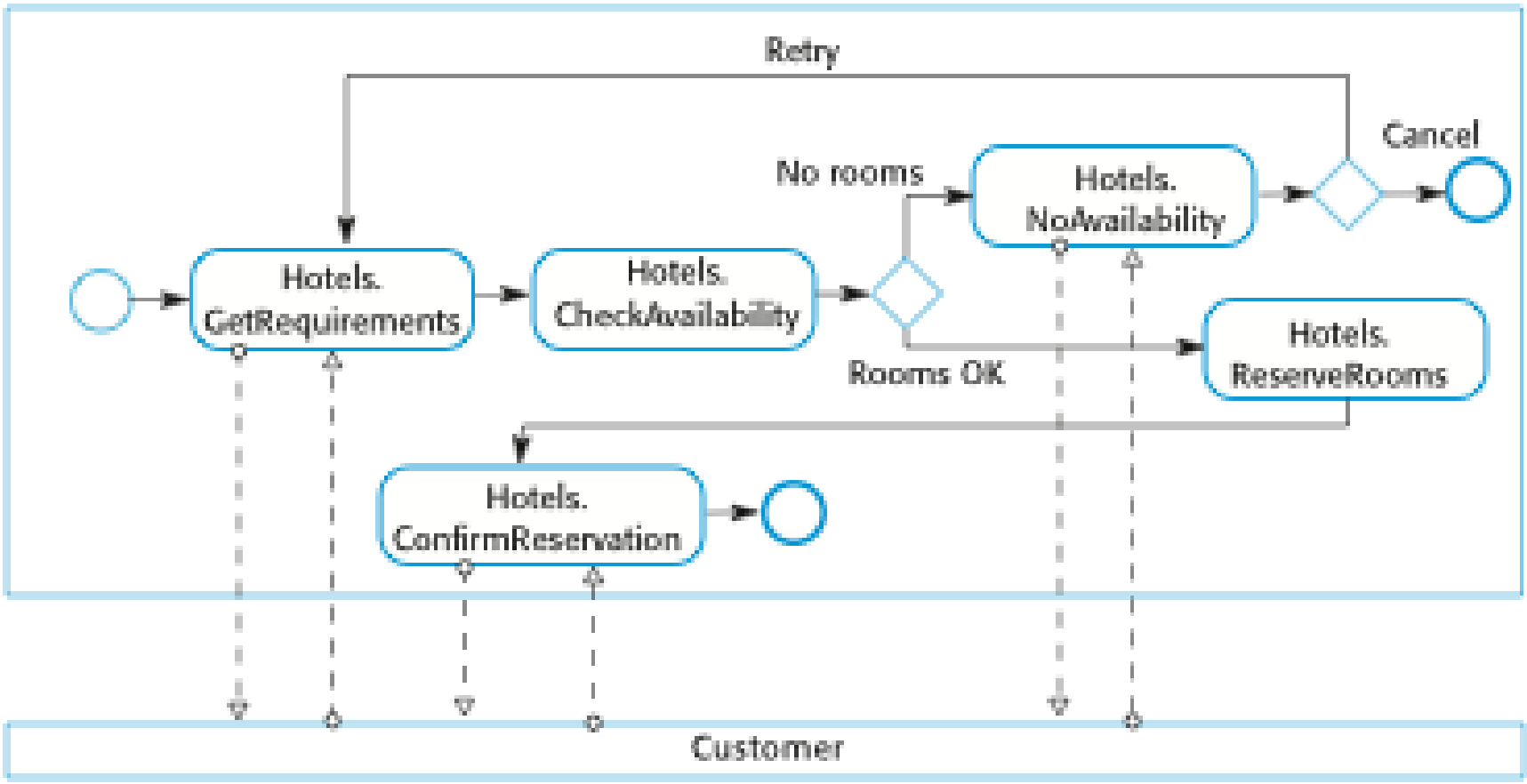
- Istniejące usługi są łączone i konfigurowane w celu stworzenia nowej, złożonej usługi lub aplikacji.
- Bardzo często jako podstawę złożenia usług wykorzystuje się przepływ prac (ang. workflow)
  - Przepływ prac to logiczna sekwencja aktywności, które wspólnie modelują spójny proces biznesowy.
    - Automatyzacja procesu biznesowego
  - Na przykład udostępnienie usług rezerwacji podróży, które umożliwiają skoordynowanie rezerwacji lotniczej, samochodowej i hotelowej.

Przepływ prac dla rezerwacji „pakietu wakacyjnego”



Konstrukcja przez kompozycję

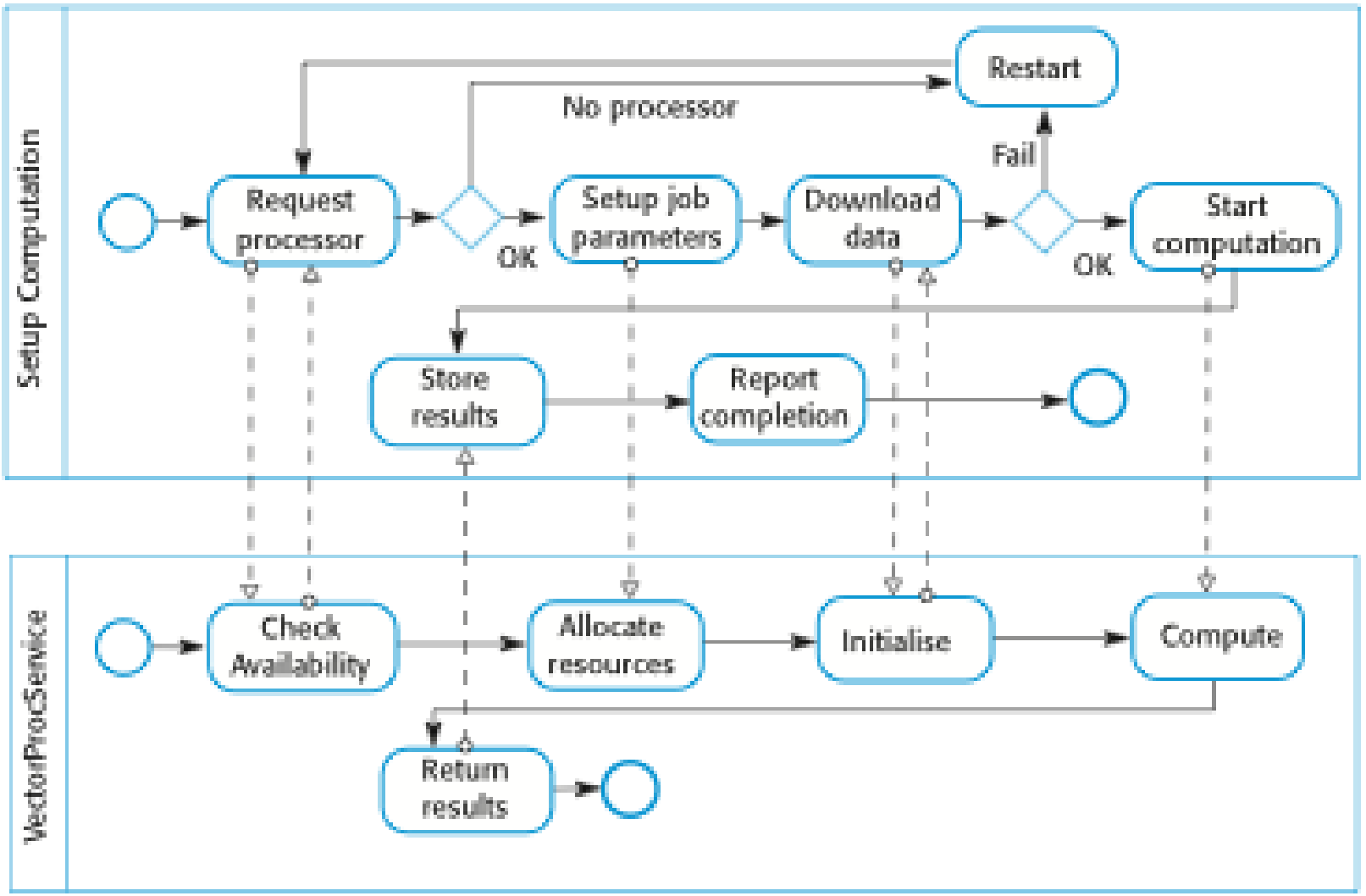
Fragment zautomatyzowanego procesu rezerwacji hotelowej



Projektowanie i implementacja przepływu prac

- WS-BPEL to, bazujący na XML, standard specyfikacji przepływu prac. Jednak opis z wykorzystaniem WS-BPEL nie nadaje się do projektowania - jest długi i nieczytelny
- Notacje takie jak BPMN, pozwalają na graficzne projektowanie – dokument WS-BPEL może być wygenerowany automatycznie. BPMN 2.0 posiada również poziom wykonywalny.
- W systemach „międzyorganizacyjnych” oddzielne przepływy prac opracowywane są dla każdej organizacji i łączone za pomocą wymiany komunikatów

Interakcja przepływów prac



Testowanie usługi

- Intencją testu jest znalezienie usterek oraz wykazanie spełniania wymagań funkcjonalnych i niefunkcjonalnych.

- Testowanie usług jest trudne ze względu na ich hermetyzację (testy czarnej skrzynki). Nie jest możliwe wykorzystanie testów bazujących na kodzie źródłowym.

Problematyka testowania usług

- Zewnętrzna usługa może zostać zmodyfikowana przez dostawcę co powoduje, że istniejące testy przestają być poprawne.
- Istnienie dynamicznych wiązań oznacza, że wykorzystywana usługa może zachowywać się zmiennie – testy aplikacji są niemiarodajne.
- Niefunkcjonalne zachowania systemu jest trudne do przewidzenia ze względu na zależność od obciążenia.
- Jeżeli za wykorzystanie usługi trzeba płacić, testowanie może być kosztowne.
- Trudne może być wywołanie akcji kompensującej zewnętrzną usługę ponieważ może ona wymagać pojawienia się błędów w innych usługach, których nie da się zasymulować.

Podsumowanie (1)

- Inżynieria oprogramowania zorientowana usługowo bazuje na założeniu, że aplikacje mogą być budowane na zasadzie kompozycji niezależnych usług. Usługi te hermetyzuję funkcjonalność wielokrotnego wykorzystania.
- Interfejsy usług definiowane są za pomocą języka WSDL. Specyfikacja interfejsu zawiera definicję typów, operacji, protokół wiązania do usługi oraz jej lokalizację.
- Usługi można podzielić na ogólne, funkcje biznesowe oraz koordynujące.

Podsumowanie (2)

- Modele procesów biznesowych definiują aktywności oraz informacje w nich wykorzystywane. Aktywności procesu biznesowego mogą być implementowane w postaci usług – powoduje to, że model procesu biznesowego może być reprezentowany jako kompozycja usług.

- Techniki testowania oprogramowania oparte na analizie kodu źródłowego nie mogą być wykorzystywane w systemach zorientowanych usługowo ponieważ tego typu systemy bazują na zewnętrznie dostarczanych usługach.