Warstwa integracji

wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.

1. **Ukrycie logiki dostępu do danych w osobnej warstwie**
2. **Oddzielenie mechanizmów trwałości od modelu obiektowego**

Pięciowarstwowy model logicznego rozdzielania zadań (wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

Warstwa klienta
Klienci aplikacji, aplety, aplikacje i inne elementy z graficznym interfejsem użytkownika

Interakcja z użytkownikiem, urządzenia i prezentacja interfejsu użytkownika

Warstwa prezentacji
Strony JSP, serwlety i inne elementy interfejsu użytkownika

Logowanie, zarządzanie sesją, tworzenie zawartości, formatowania i dostarczanie

Warstwa biznesowa
Komponenty EJB i inne obiekty biznesowe

Logika biznesowa, transakcje, dane i usługi

Warstwa integracji
JMS, JDBC, konektory i połączenia z systemami zewnetrznymi

Adaptery zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania

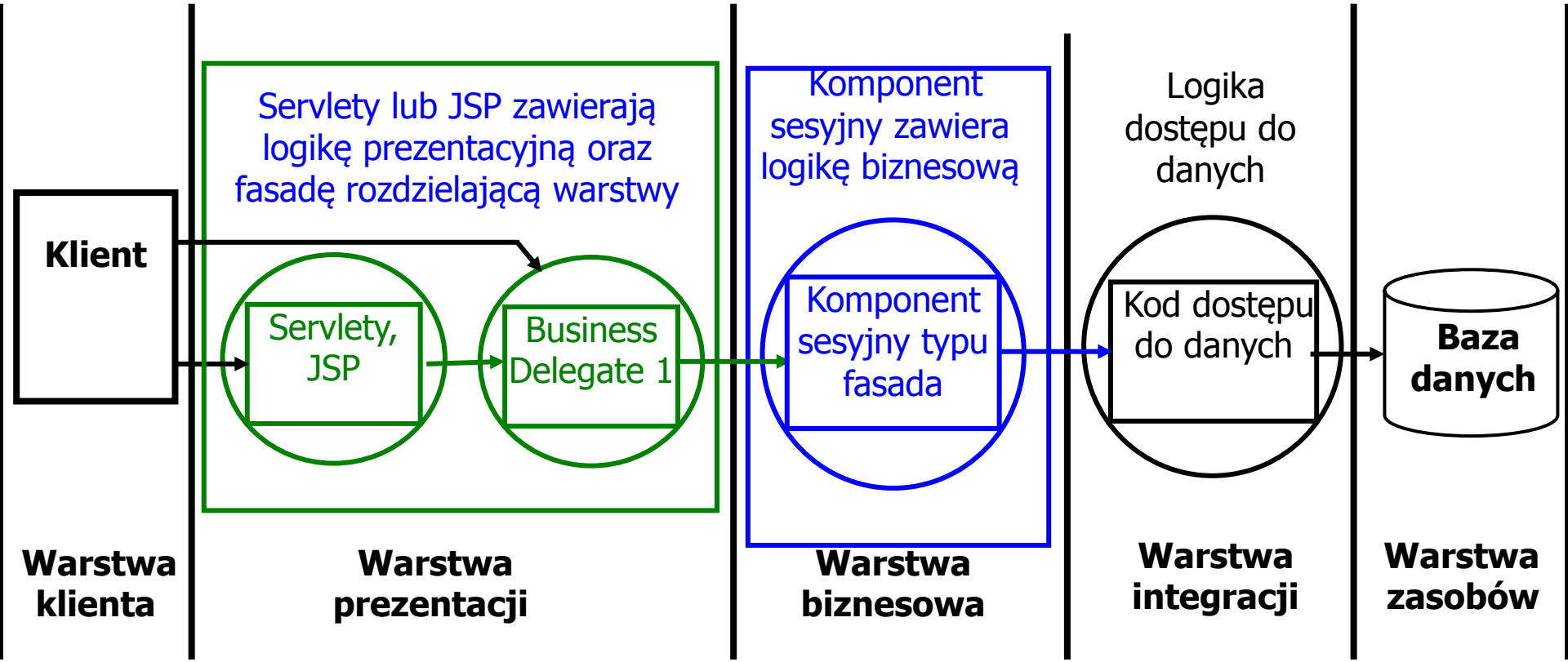
Warstwa zasobów
Bazy danych, systemy zewnętrzne i pozostałe zasoby

Zasoby, dane i usługi zewnętrzne

Problem 1 – ukrycie logiki dostępu do danych w osobnej warstwie

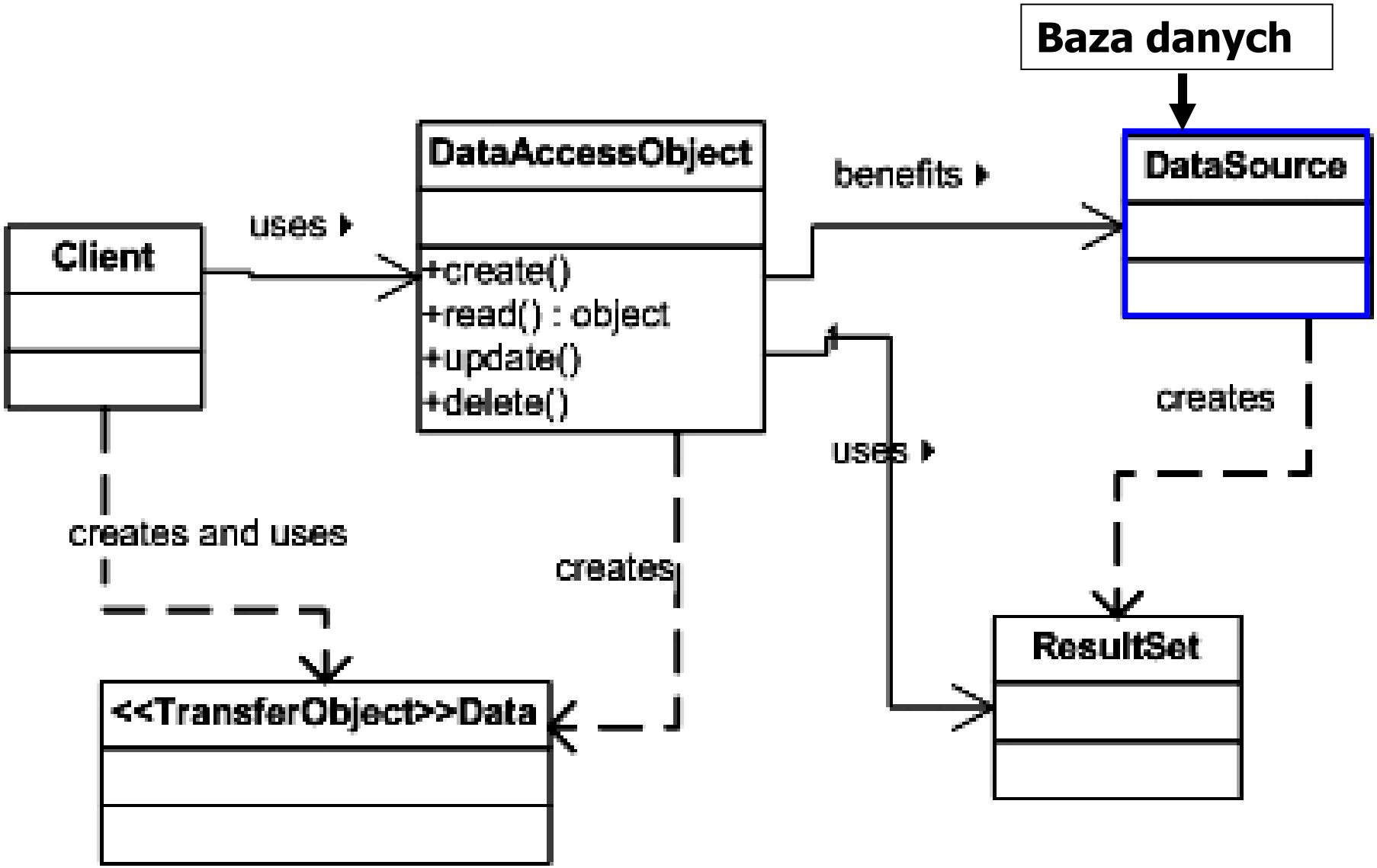
1. Aplikacje o mniejszych rozmiarach nie używają obiektów typu „entity”, natomiast w fasadowych obiektach sesyjnych zawarta jest logika biznesowa, a przetwarzanie danych odbywa się **bezpośrednio w trwałym magazynie**.
2. Większość aplikacji biznesowych używa jako **trwałych magazynów** relacyjnych baz danych (RDBMS), zewnętrznych systemów mainframe, repozytoriów, obiektowych baz danych (OODB), zwykłych plików, usług zewnętrznych systemów np.. B2B lub usług kart kredytowych. Każdy z tych systemów używa innych mechanizmów dostępu obsługiwanymi za pomocą API oraz posiada inną funkcjonalność

Architektura systemu informatycznego



Wymagania mechanizmu trwałości znajdującego się w osobnej warstwie :

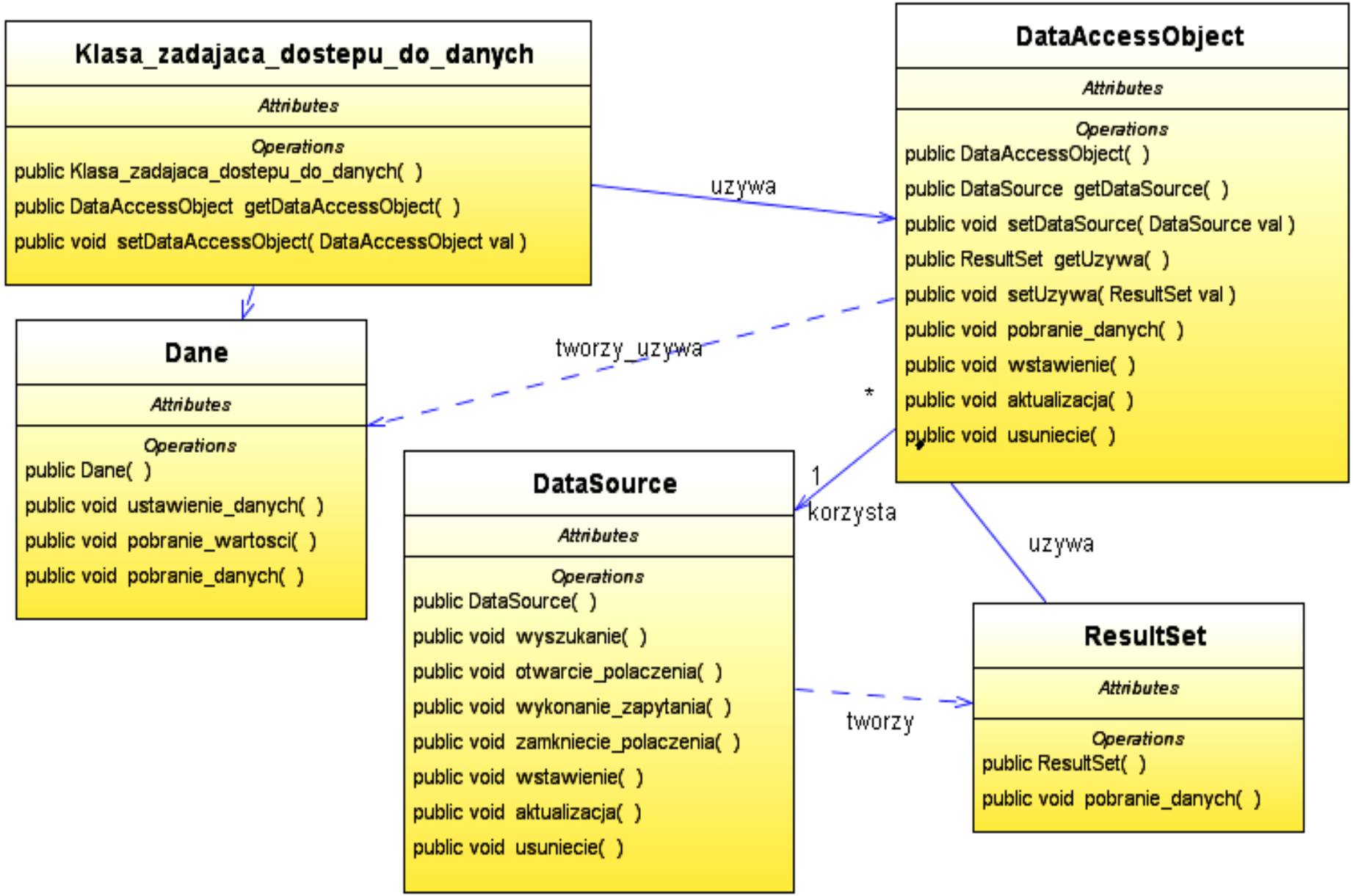
1. Należy zaimplementować mechanizmy dostępu do danych, aby pobierać i modyfikować dane znajdujące się w trwałym magazynie
2. Należy oddzielić implementacje trwałego magazynu od reszty systemu
3. Należy stworzyć jednolity interfejs dostępu do danych, znajdujących się w różnych źródłach np.. w RDBMS, LDAP, OODB, repozytoriach XML, zwykłych plikach
4. Należy dokonać organizacji logiki dostępu do danych i ukryć w jednym miejscu specyficzne funkcje w celu osiągnięcia lepszej przenośności i ułatwienia konserwacji kodu


Baza danych
DataAccessObject
+create()
+read() : object
+update()
+delete()
DataSource
benefits
Client
uses
creates
uses
creates and uses
creates
ResultSet
<<TransferObject>>Data

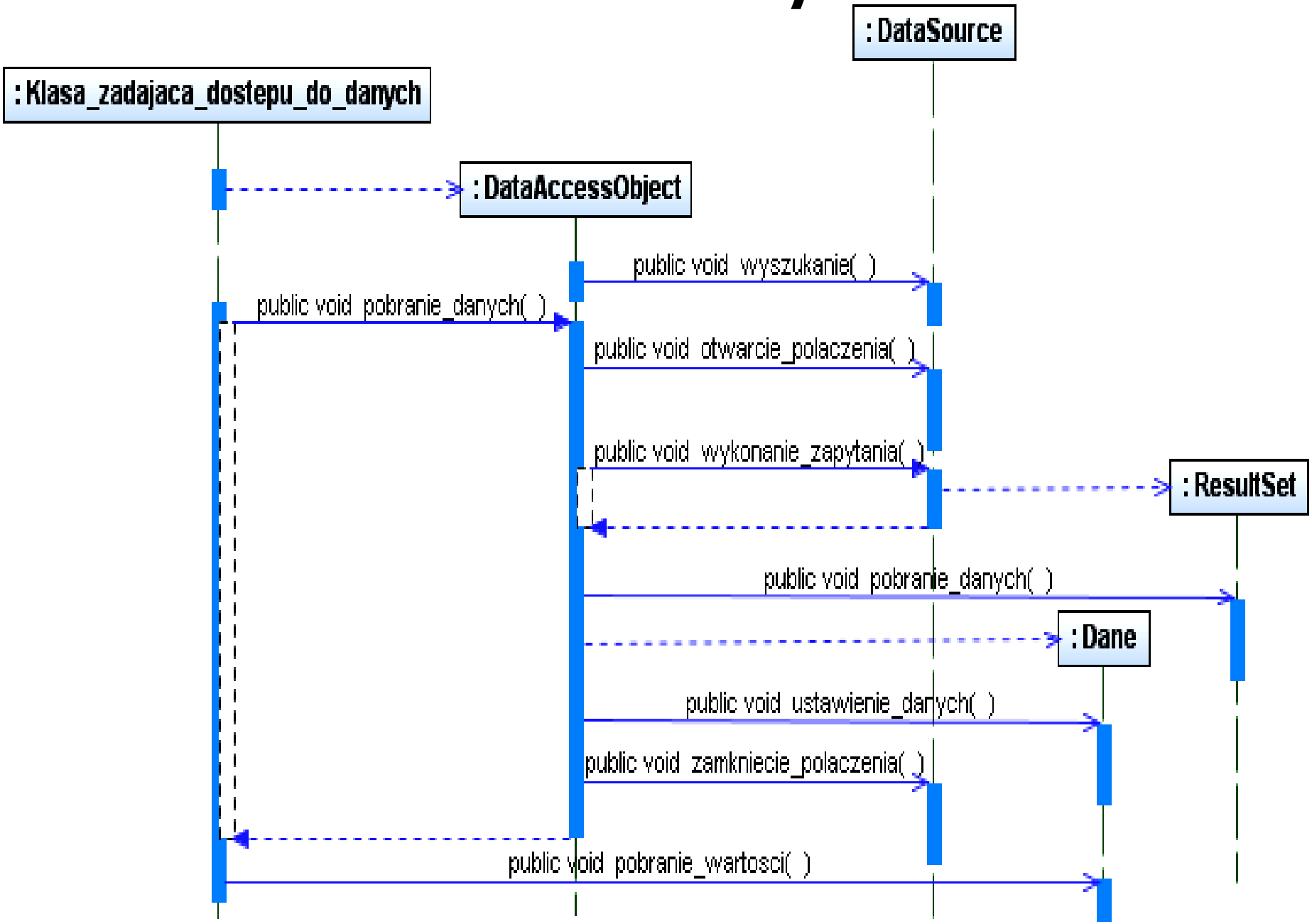
Obiekty warstwy integracji

- **Client** – jest obiektem żądającym dostępu do danych w celu pobrania lub zapisania danych. Klientem może być obiekt biznesowy (obiekt modelu obiektowego), komponent **SessionFacade, Application Service** lub dowolny inny obiekt pomocniczy wymagający dostępu do trwałych danych
- **DataAccessObject** – jest głównym elementem wzorca. Ukrywa on przez klientem rzeczywistą implementację dostępu do danych, aby zapewnić jednakowy dostęp, niezależny od typu **DataSource.** Obiekt ten implementuje operacje **CRUD** (Create, Read, Update, Delete).
- **DataSource** - dowolna usługa zarządzająca danymi. Może to być relacyjna lub obiektowa baza danych itp.
- **ResultSet** – reprezentuje wynik zapytania. Dla sterowników JDBC jest to instancja typu java.sql.ResultSet
- **Data** – reprezentuje obiekt transferowy, wykorzystywany do przenoszenia danych.

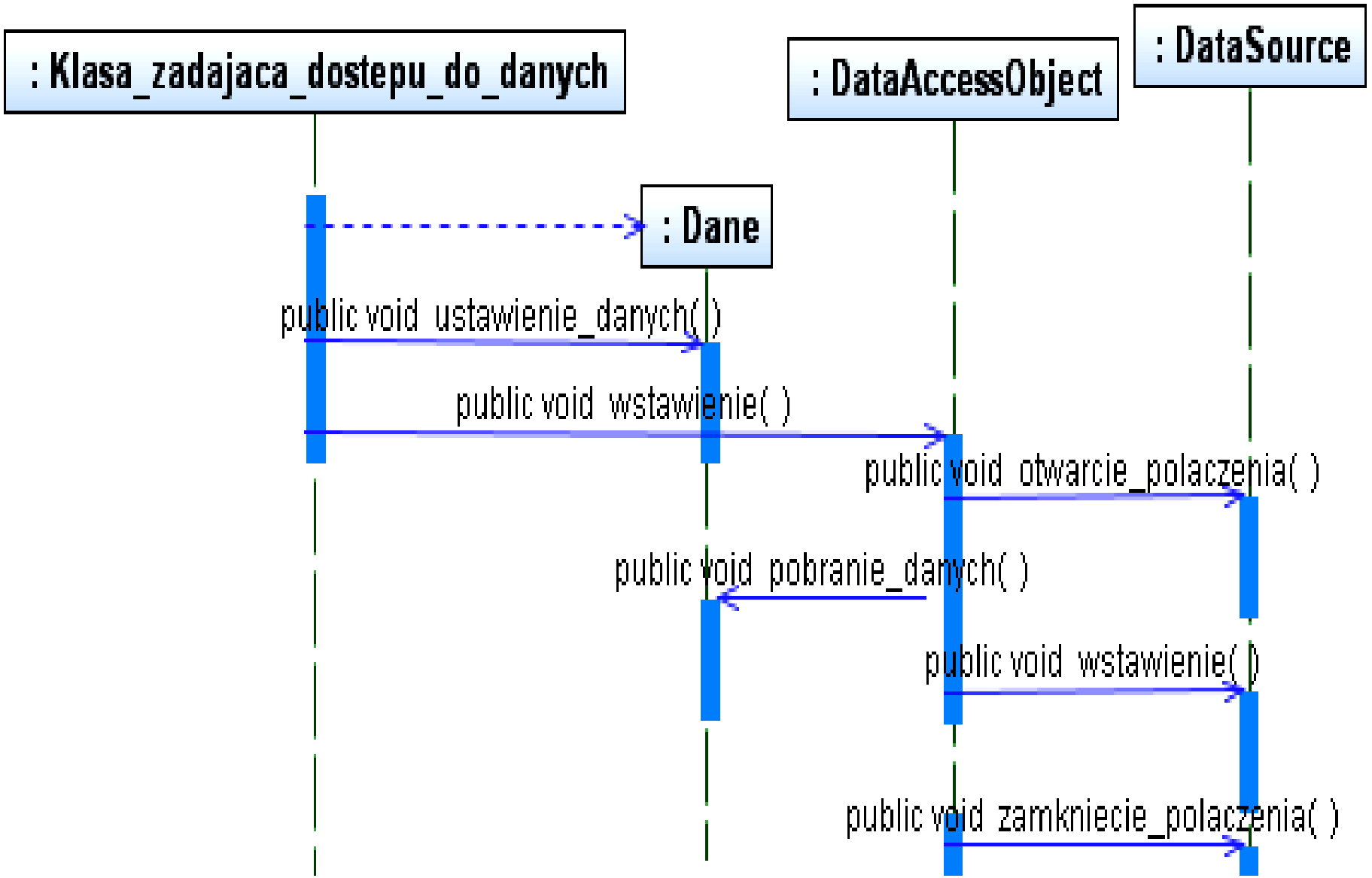
Strategia własnego obiektu dostępu do danych



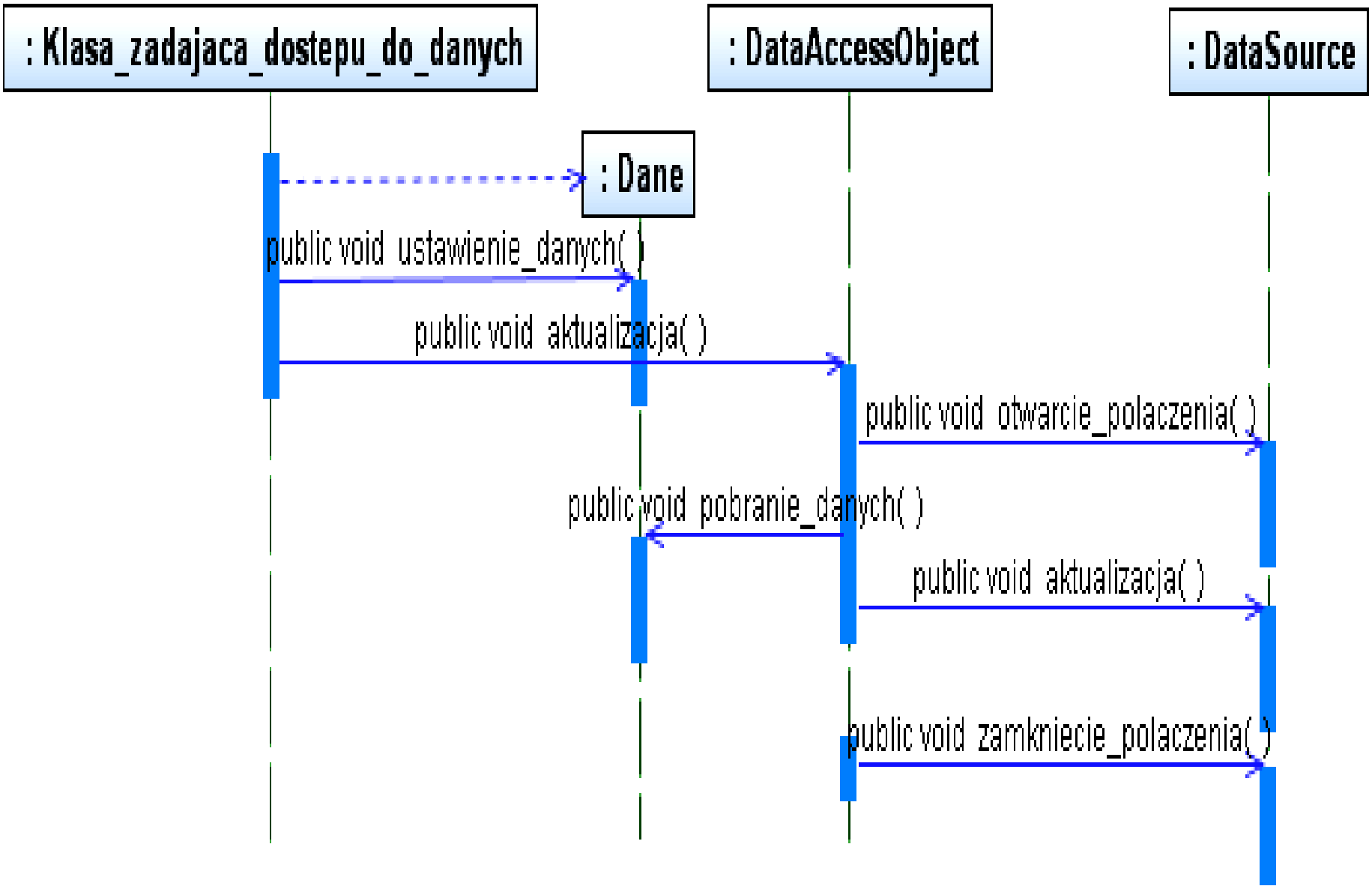
Pobranie danych



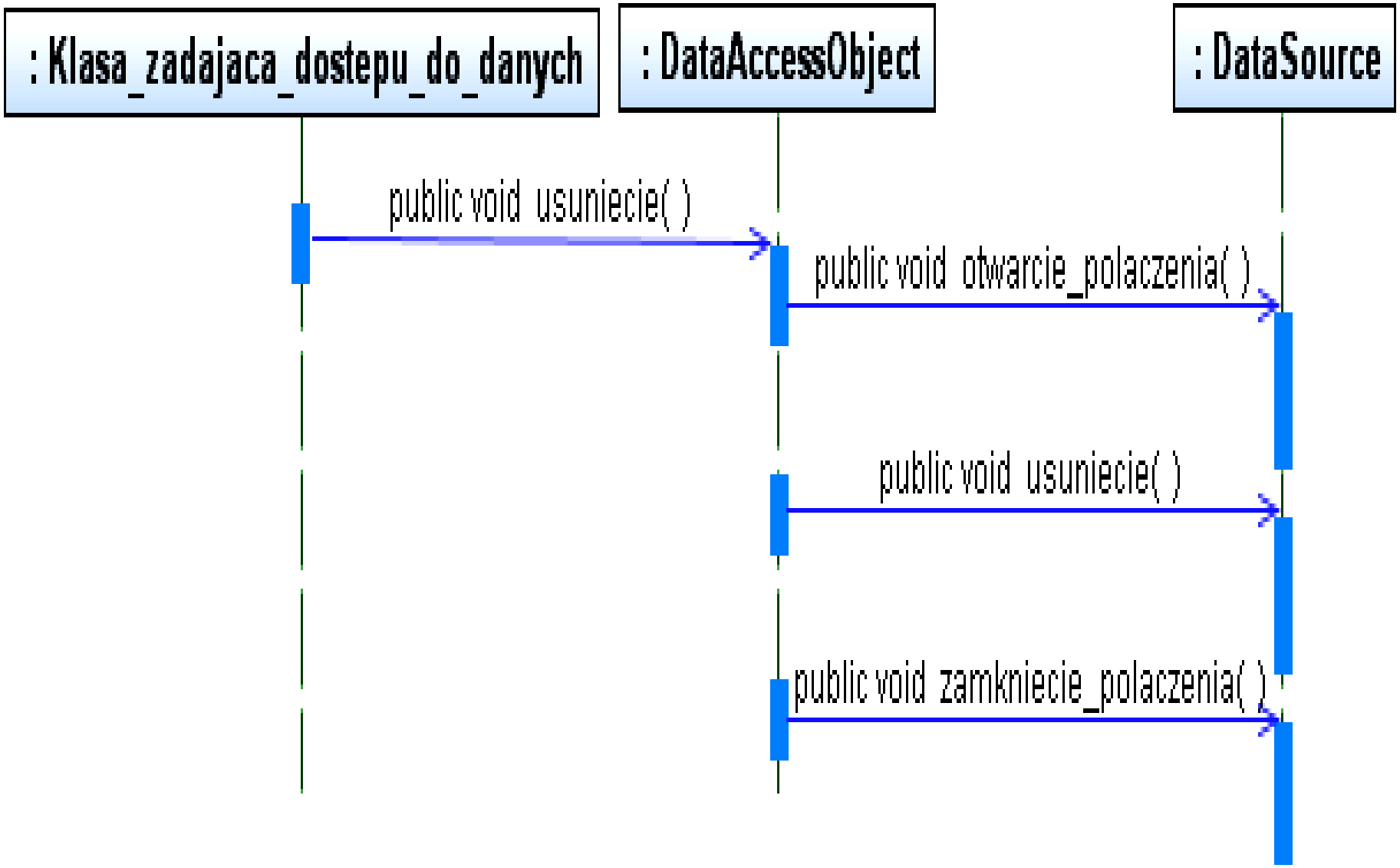
Wstawienie danych



Aktualizacja danych



Usuwanie danych



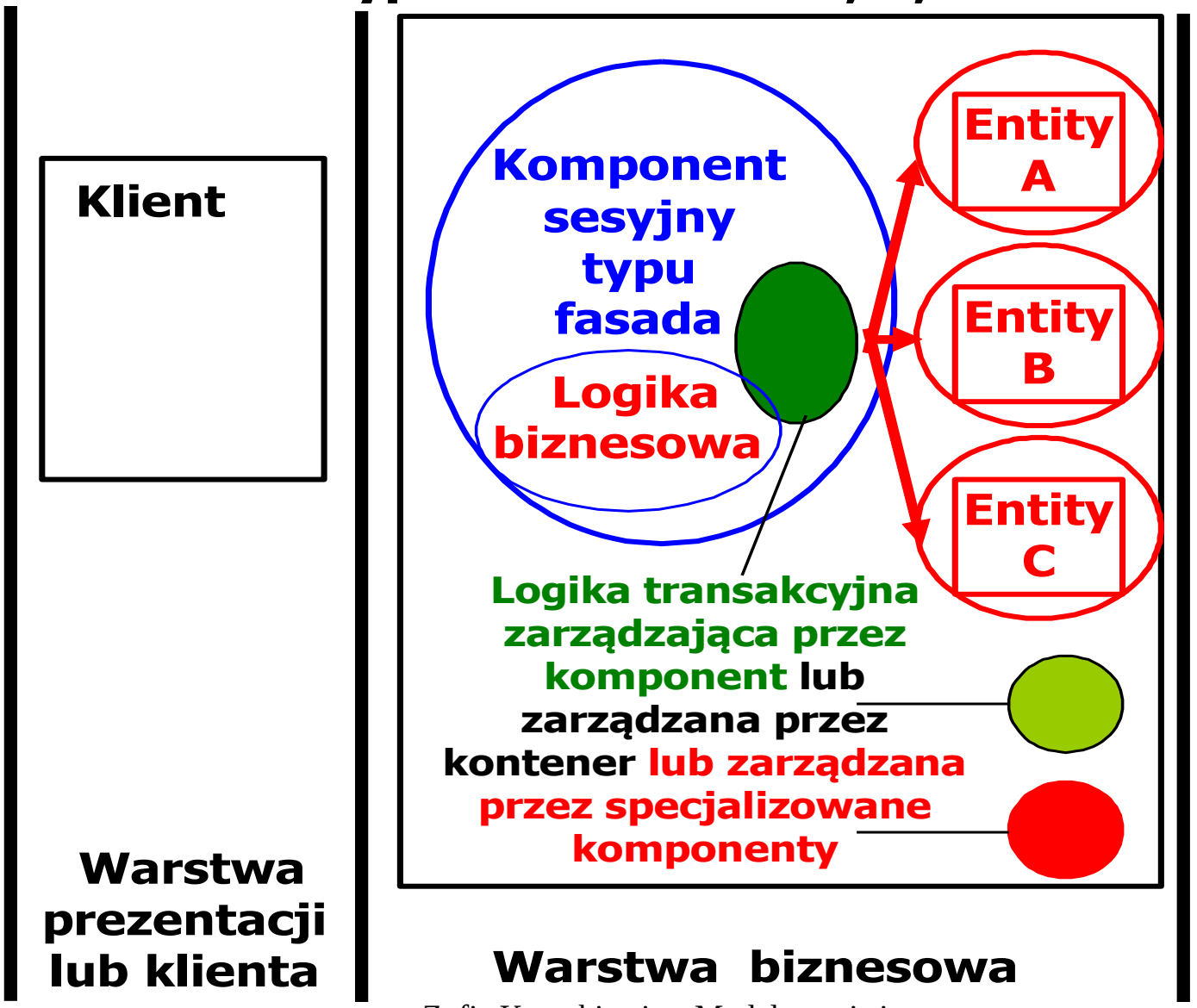
Problem 2 – oddzielenie mechanizmów trwałości od modelu obiektowego

1. Wiele systemów posiada złożony model obiektowy zbudowany ze zwykłych obiektów typu „entity” lub komponentów typu „Entity”). Wymaga on wyrafinowanych strategii utrwalania w **trwałych magazynach.**
2. Alternatywna 1 - kontenery warstw biznesowych zarządzają trwałością obiektów typu „entity”
3. Alternatywna 2 - obiekty typu „entity” zarządzają trwałością , co może być wykorzystane w implementacji trwałości modelu obiektowego
4. **Alternatywna 3** – uruchamianie aplikacji w warstwie prezentacji i oddzielenie mechanizmów trwałości od modelu obiektowego. Modele obiektowe można oprzeć na tzw. niewidocznej trwałości, czyli obiekty biznesowe modelu obiektowego nie odpowiadają za implementację mechanizmów trwałości.

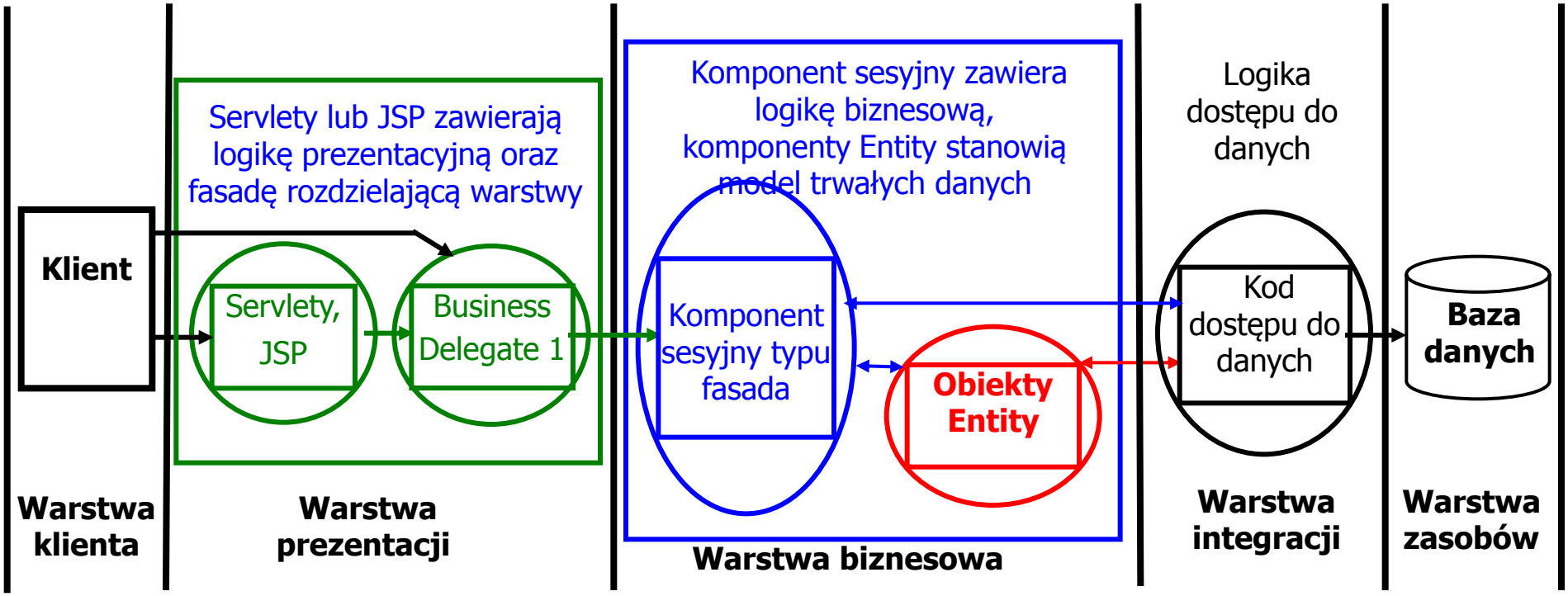
Model warstwy biznesowej odwołującej się do pięciu typów trwałości

1. Brak modelu obiektowego (brak obiektów typu „entity”) - rozwiązanie problemu 1
2. Mechanizm trwałości jest zaimplementowany w kontenerach, w których znajdują się obiekty typu „entity” (strategia CMP – *Container-managed Persistence*)- brak dziedziczenia w modelu obiektowym
3. Mechanizm trwałości jest zaimplementowany w obiektach typu „entity” (strategia BMP - *Bean-managed Persistence*) – kod trwałości wymieszany z modelem obiektowym)- brak dziedziczenia w modelu obiektowym
4. Mechanizm trwałości jest umieszczony w modelu obiektów typu „entity” – kod trwałości wymieszany z modelem obiektowym
5. Mechanizm trwałości jest oddzielony od modelu obiektów typu „entity” opartego na wszystkich wzorcach obiektowości

Model warstwy biznesowej odwołującej się do typów trwałości: 2, 3, 5

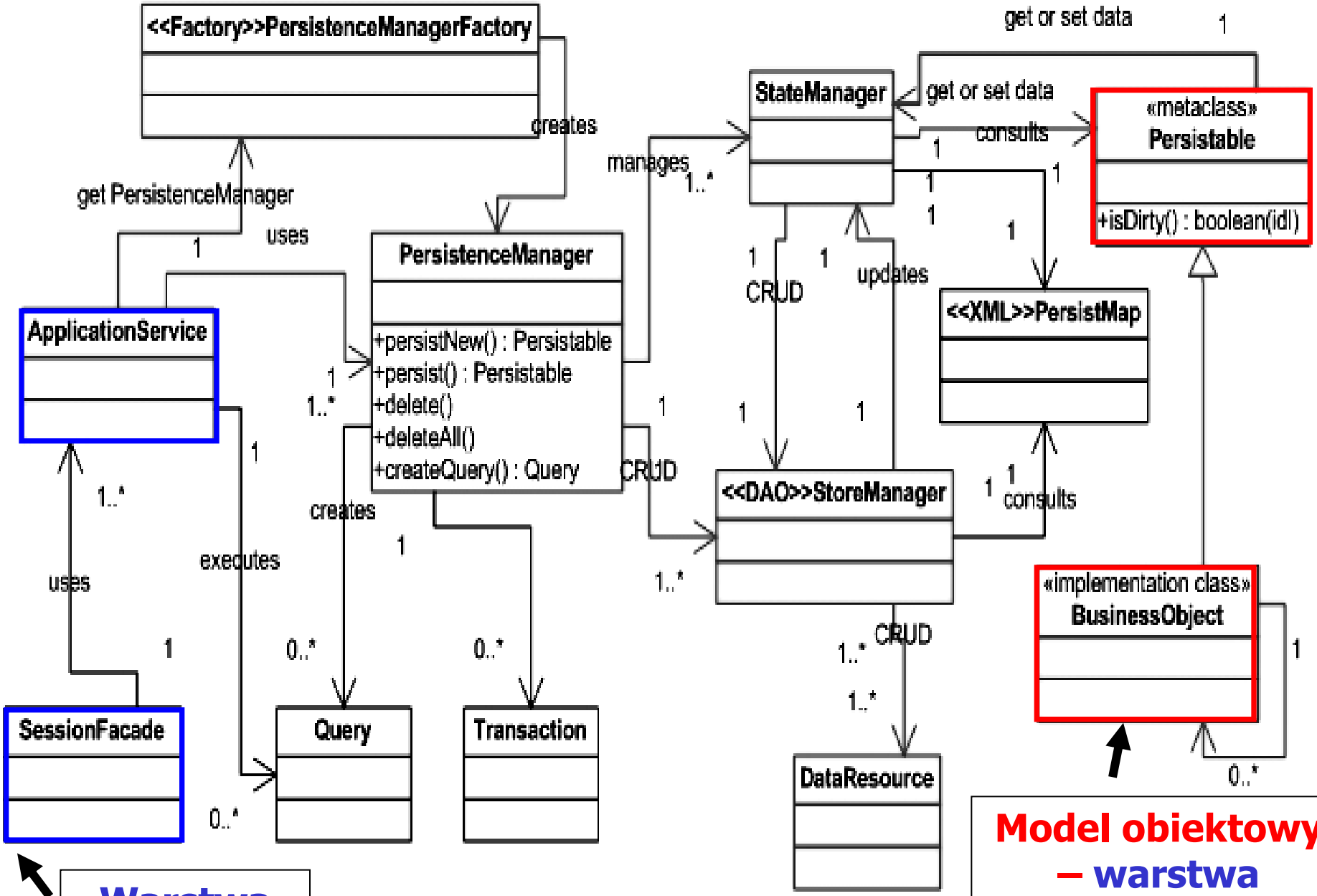


Architektura 5-warstwowego systemu informatycznego



Wymagania mechanizmu trwałości oddzielonego od modelu obiektów typu „entity”:

1. Należy uniknąć implementacji mechanizmów trwałości w obiektach biznesowych typu „entity”.
2. Należy zrezygnować z obiektów typu „entity”, które korzystają z mechanizmów trwałości kontenera.
3. Aplikacja może działać w kontenerze web.
4. Model obiektowy używa dziedziczenia i złożonych relacji.
5. Mechanizm trwałości można rozwiązać dwoma sposobami:
   - wykonać własny szkielet trwałości
   - lub zastosować gotowe rozwiązania oparte na JDO lub własnych rozwiązaniach O-R



Warstwa biznesowa

Model obiektowy – warstwa biznesowa

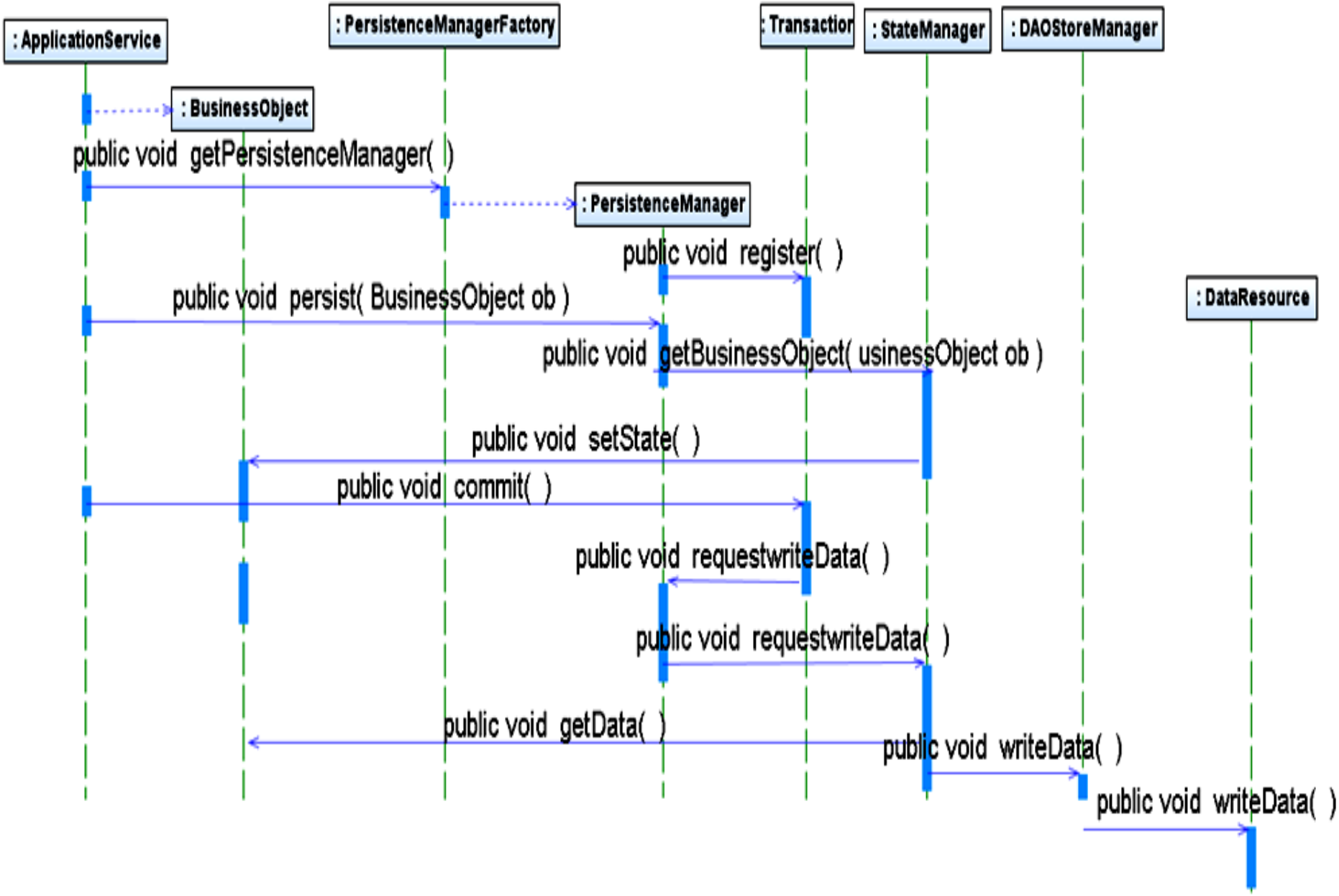
Opis głównych obiektów warstwy integracji

- **Application Service** – obiekt usług wykorzystujący obiektu modelu obiektowego
- **Persistable** – interfejs lub klasa bazowa dla wszystkich trwałych obiektów biznesowych
- **PersitenceManagerFactory** – tworzy obiekty **PersistenceManager** i zarządza nimi
- **PersistenceManager** – zarządza trwałością i zapytaniami modelu obiektowego. Obiekt **PersistenceManager** zarządza obiektami modelu obiektowego za pośrednictwem obiektu **StateManager**
- **StateManager -** zarządza stanem obiektów modelu obiektowego. Wymusza on transakcyjny zapis i pobieranie stanu obiektów z **DataResource**
- **StoreManager (DAO)** – współdziała z **DataResource**, w celu przeprowadzania operacji CRUD (Create, Read, Update, Delete). Obiekt **StoreManager** jest obiektem DAO i zawiera wszystkie mechanizmy dostępu do danych.
- **DataResource** – dowolna usługa zarządzająca danymi. Może to być relacyjna lub obiektowa baza danych itp..

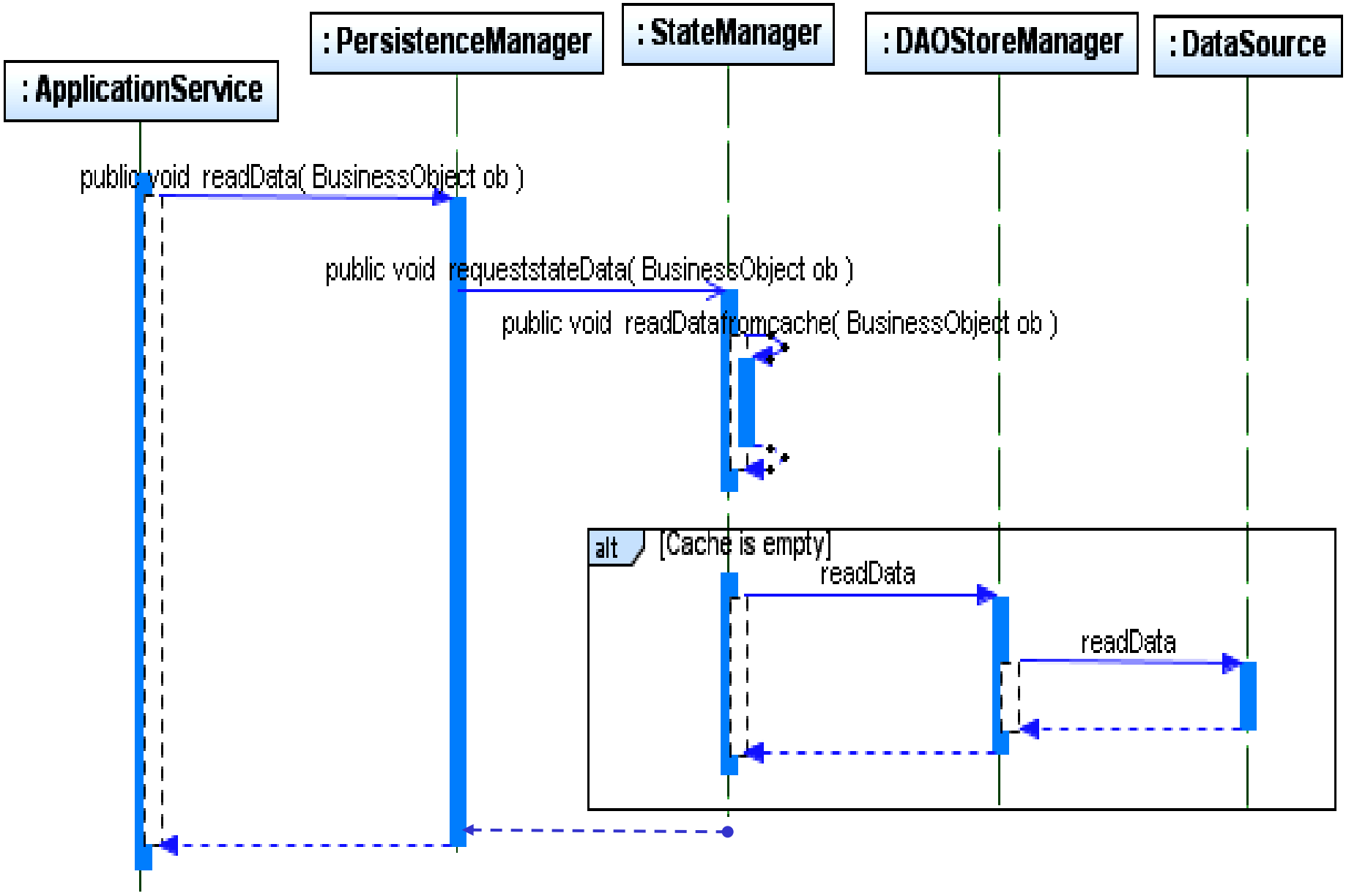
Dodatkowe obiekty warstwy integracji

- **SessionFacade** – punkt dostępu do warstwy biznesowej. Współpracuje ona z jednym lub kilkoma obiektami ApplicatopnService
- **PersistMap** – zawiera definicje realacji miedzy obiektami , a także odwzorowanie między trwałymi obiektami a źródłem danych
- **Transaction** - jest związany z obiektem **PersistenceManager**. Służy on do definiowania reguł transakcji oraz do samodzielnego zarządzania transakcjami w środowiskach bez takiego wsparcia
- **Query** – hermetyzuje zapytanie, kryteria filtrowania, porządkowania i deklaracji parametrów

Tworzenie i utrwalanie obiektu biznesowego



Pobieranie danych



Tworzenie i wykonanie zapytania