Paradygmaty Programowania
Programowanie zdarzeniowe

dr inż. Marcin Bączyk

Wykład 9

9 maja 2022

Treść dzisiejszego wykładu

- Programowanie zdarzeniowe:
  - zdarzenie
  - dyspozytor
  - pętla zdarzeń
- Systemy przesyłania wiadomości i ich topologie:
  - dyspozytor,
  - mediator
- Programowanie graficznych interfejsów użytkownika
- Wzorzec projektowy obserwator

Programowanie zdarzeniowe

Programowanie Zdarzeniowe (ang. *Event-driven programming*)

Metodyka wykorzystująca założenie, że w danym systemie istnieje ograniczona liczba **zdarzeń**, które mogą wystąpić i które mogą być **obsłużone**, tzn. na które aplikacja powinna w jakiś sposób zareagować. Do programisty należy wybór zdarzeń, które wymagają obsłużenia.

W programowaniu zdarzeniowym występuje niedeterministyczny przepływ sterowania.

Zdarzenie (ang. *Event*)

Obiekt utworzony przez źródło zdarzenia. Obiekt zdarzenia zawiera wszystkie informacje niezbędne do jego obsłużenia.

Celem utworzenia obiektu zdarzenia jest enkapsulacja jego parametrów i umożliwienie przekazania ich do obiektu lub funkcji, w którym dane zdarzenie jest obsługiwane.

Występuje tu pewna odległa analogia do wzorca Polecenie.

Programowanie zdarzeniowe

Najczęściej spotykane zdarzenia są generowane przez urządzenia wejścia-wyjścia lub elementy graficznego interfejsu użytkownika (GUI):

- wciśnięcie przycisku – elementu GUI,
- wciśnięcie klawisza na klawiaturze,
- kliknięcie lewym lub prawy przyciskiem myszy,
- ruch myszą w określony sposób.

Zdarzenia mogą być związane z obsługą wszelkiego rodzaju komunikacji:

- otrzymanie komunikatu od innych programów,
- otrzymanie wyniku pomiaru.

Programista może też ustawić zdarzenia, które będą wygenerowane wyznaczonej chwili czasowej.

Programowanie zdarzeniowe

Programowanie zdarzeniowe dotyczy również obsługi przerwań sprzętowych (oraz niekiedy programowych) programów uruchamianych bezpośrednio na procesorze lub mikrokontrolerze (bez pośredniczenia systemu operacyjnego). Zdarzenia w takim systemie nazywane są **przerwaniami** i mogą być zgłoszone na przykład przez:

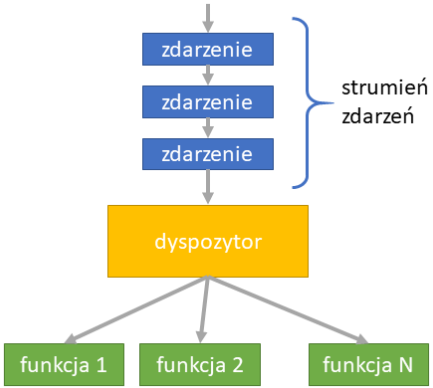
- kontroler pamięci,
- kontroler magistrali,
- licznik zegara.

W takim systemie dla konkretnych przerwań (zdarzeń) można zdefiniować procedury ich obsługi.

Programowanie zdarzeniowe

System obsługi zdarzeń składa się ze:

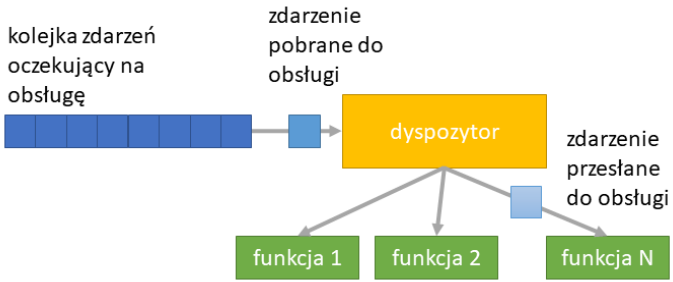
- strumienia napływających zdarzeń,
- dyspozytora zdarzeń,
- zestawu funkcji obsługujących zdarzenia.



Dyspozytor (ang. *dispatcher*)

Zadaniem dyspozytora jest dostarczanie poszczególnych zdarzeń do konkretnej funkcji obsługującej w zależności od typu zdarzenia.

Dyspozytor to nieskończona pętla, w której przetwarzany jest wejściowy strumień zdarzeń. Zdarzenia kolejno pobierane są z pętli zdarzeń, dekodowane i wysyłane do obsługujących je funkcji – odpowiednich do rodzaju zdarzenia.



Pętla (kolejka) zdarzeń

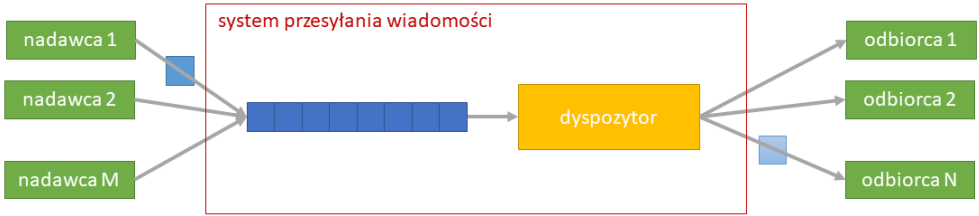
W niektórych przypadkach dyspozytor oraz procedury obsługi zdarzeń mogą nie być w stanie obsłużyć zdarzeń dostatecznie szybko. W takich przypadkach rozwiązaniem jest buforowanie strumienia zdarzeń wejściowych w postaci kolejki.

Wydarzenia są dodawane na koniec kolejki tak szybko, jak przybywają, natomiast dyspozytor pobiera je z początku kolejki, przekazując poszczególnych funkcjom obsługującym.

Aplikacje GUI zazwyczaj zawierają kolejkę zdarzeń. Niektóre istotne zdarzenia, takie jak kliknięcia myszą, mogą wymagać złożonej obsługi. Podczas ich obsługi inne zdarzenia, takie jak ruch myszy, mogą gromadzić się w buforze. Następnie od dyspozytora zależy, czy zdarzenia te zostaną obsłużone, czy pominięte.

Systemy rozsyłania wiadomości

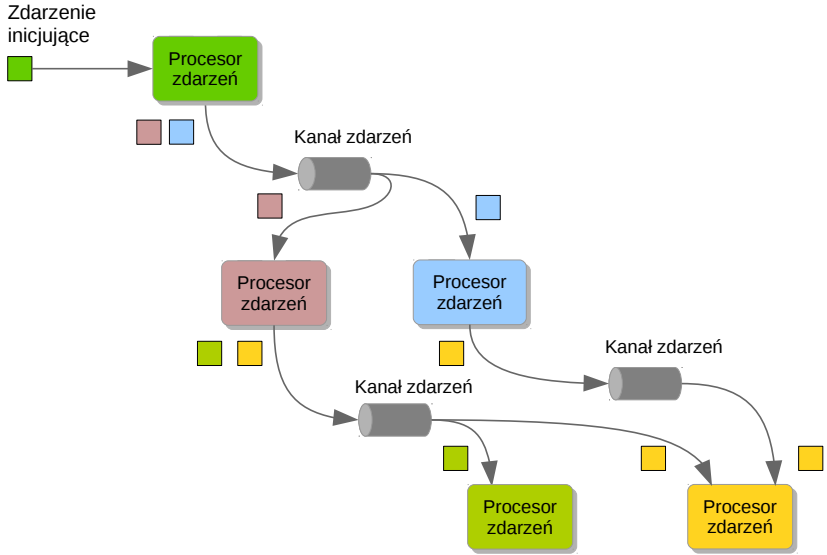
Systemy rozsyłania wiadomości reprezentują skrajną wersję wzorca obsługi zdarzeń. Celem systemu jest odbieranie zdarzeń (komunikatów) od generatorów zdarzeń (nadawców) i przesyłanie ich do funkcji obsługujących (odbiorców) w sytuacjach, gdy nadawcy i odbiorcy znajdują się w różnych lokalizacjach fizycznych lub działają na różnych platformach.



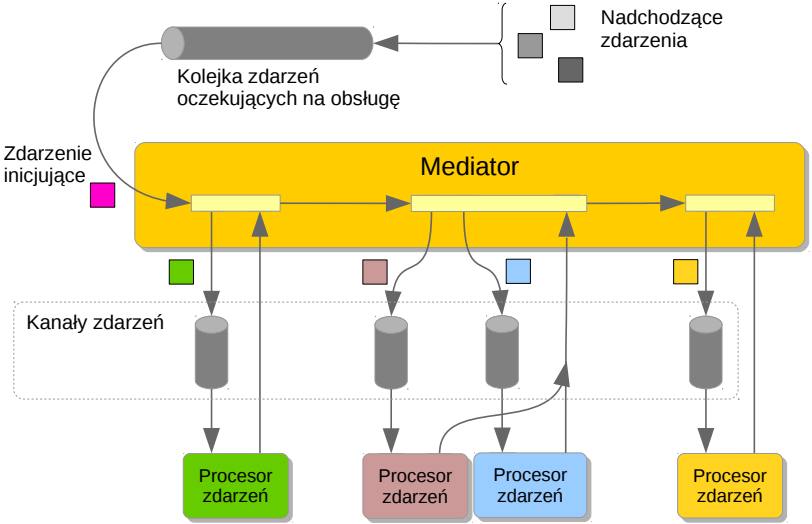
System rozsyłania wiadomości może być zorganizowany w dwóch topologiach:

- **Brokera**. Wykorzystywana, gdy niezbędny jest wysoki stopień responsywności i dynamiczna kontrola nad przetwarzaniem zdarzenia.
- **Mediatora**. Wykorzystywana, gdy proces przetwarzania zdarzenia jest wieloetapowy i niezbędna jest kontrola nad przebiegiem.

System rozsyłania wiadomości – topologia brokera

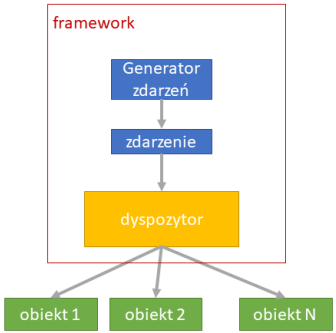


System rozsyłania wiadomości – topologia mediatora



Programowanie zdarzeniowe a programowanie obiektowe

Dzięki programowaniu zorientowanemu obiektowo stosunkowo łatwo jest opracować uogólnione klasy wielokrotnego użytku. Jest to jedna z zalet tej technologii. Uogólnione klasy wielokrotnego użytku stanowią podstawę do tworzenia własnych implementacji, które następnie włączane są (*plug-in*) w większą strukturę. Użytkownik, tworząc własną klasę, implementuje jedynie mały fragment kodu obsługujący konkretne zdarzenie.
System obsługi zdarzeń używa używa obiektów klas zdefiniowanych przez programistę, tak jakby były one obiektami klas dostarczonych przez twórców danego systemu. Tworzone przez programistę obiekty wstawiane są w miejsce, w którym powinny się znaleźć obiekty obsługujące dane zdarzenie.



Tworzenie graficznych interfejsów użytkownika

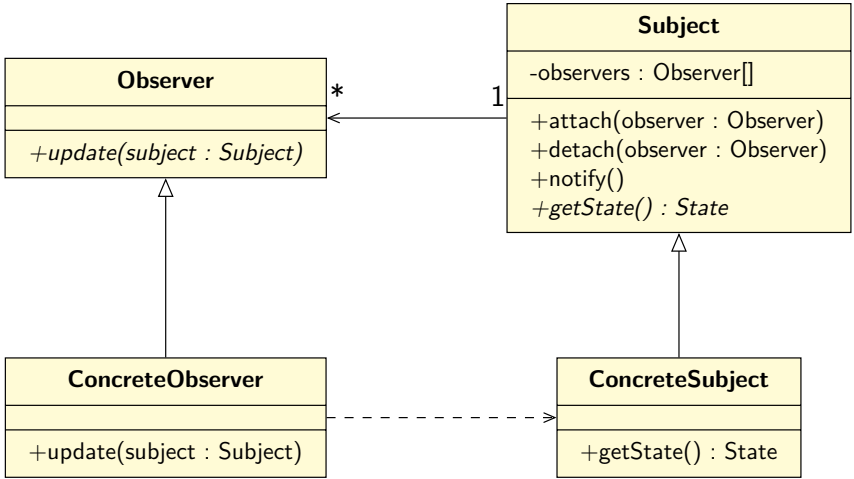
- W programowaniu GUI najważniejsze jest zdefiniowanie **obiektu** obsługującego dane zdarzenie.
- Należy określić wygląd każdego aspektu projektowanego interfejsu.
- Istnieje wiele rodzajów zdarzeń, które muszą być obsługiwane przez GUI.
- Istnieje wiele modyfikatorów zachowania standardowych akcji (wciśnięcie klawisza Ctrl, Alt lub Shift).
- Narzędzia do tworzenia graficznych interfejsów użytkownika dostarczane są jako frameworki, które z jednej strony zdejmują dużą część pracy z programisty, a z drugiej strony trzeba się ich nauczyć.
- Często wykorzystywany jest wzorzec **obserwatora**, który trzeba zrozumieć i wiedzieć, jakie są jego ograniczenia.

Wzorzec projektowy «obserwator»

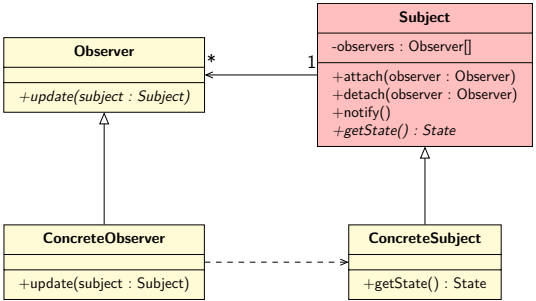
Przykład

Obiekt reprezentujący zegar informuje inne obiekty o upływającym czasie w ustalonych interwałach. Informowane obiekty mogą wyświetlać czas lub przetwarzać i przekazywać informację dalej. Na przykład obiekt reprezentujący alarm może informować od osiągnięciu danej chwili lub o upłynięciu ustalonego czasu.

Wzorzec projektowy «obserwator»



Wzorzec projektowy «obserwator»

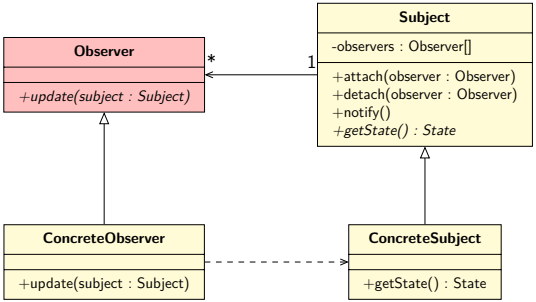


Obserwowany (podmiot)

- Zapewnia interfejs pozwalających na dołączenie lub odłączenie dowolnej liczby **Obserwatorów**.
- Zna swoich **Obserwatorów**, choć nie wie, do jakich klas konkretnych należą.

Obserwowany może zapewniać osobną metodę pozwalającą na uzyskanie informacji o stanie lub przekazać stan w wywołaniu metody powiadamiającej **Obserwujących**

Wzorzec projektowy «obserwator»

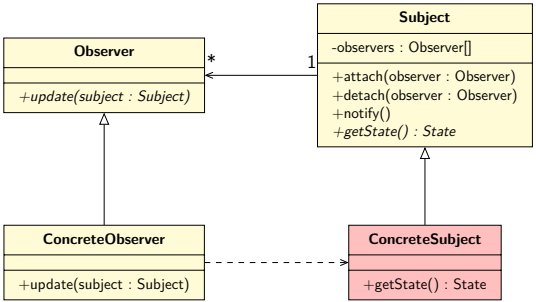


Obserwator

- Definiuje interfejs powiadamiania dla obiektów, które powinny być informowane o zmianach (konkretnych obserwatorów).

Interfejs powiadamiania może przyjmować komplet informacji niezbędnych do aktualizacji **Obserwatora** lub referencję do obiektu, który obserwuje, w celu pozyskania dalszych informacji. Metoda powiadamiania może też nie przekazywać żadnych informacji, a być jedynie oznaką tego, że coś zostało zmienione. W takim przypadku **Obserwator** niezbędne informacje do aktualizacji swojego stanu musi pozyskać w inny sposób.

Wzorzec projektowy «obserwator»

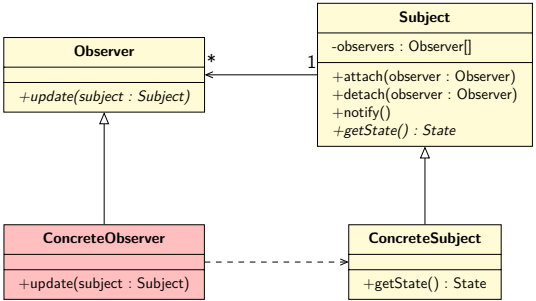


KonkretnyObserwowany

- Przechowuje stan.
- W razie zmiany stanu wysyła do swoich **Obserwatorów** powiadomienia pozwalające im uaktualnić ich stan.

KonkretnyObserwowany w razie potrzeby może implementować interfejsy wielu **Obserwowanych**.

Wzorzec projektowy «obserwator»



KonkretnyObserwator

- Przechowuje stan zgodny ze stanem obserwowanego obiektu.
- Implementuje interfejs **Obserwatora** w celu zachowania spójności swojego stanu ze stanem **Obserwatora**.

KonkretnyObserwator może nie mieć świadomości, który konkretnie obiekt obserwuje. **KonkretnyObserwator** w razie potrzeby implementuje mechanizm komunikacji z **Obserowowanym** w celu uzyskania niezbędnych informacji.

Wzorzec projektowy «obserwator»

KonkretnyObserwowany poprzez **Obserwowanego** zawsze powiadamia swoich **Obserwatorów**, gdy wystąpi zmiana jego stanu. Po otrzymaniu powiadomienia **KonkretnyObserwator** aktalizuje swój stan. W zależności od interfejsu powiadamiania, konieczne może być dodatkowa komunikacja w celu uzupełnienia informacji.

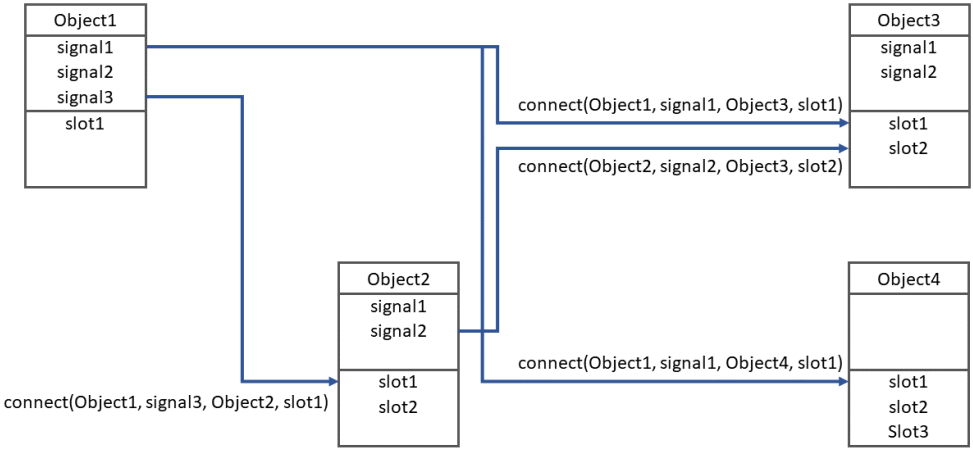
Wzorzec projektowy «obserwator»

Obszar zastosowań

Stosuj wzorzec obserwatora, gdy:

- Jakaś abstrakcja ma dwa aspekty, jeden zależy od drugiego, a enkapsulacja tych aspektów w oddzielnych obiektach umożliwia niezależne zmienianie i używanie tych obiektów.
- Zmiana jednego obiektu wymaga zmiany innych i nie wiadomo, ile obiektów trzeba zmienić.
- Obiekt powinien być w stanie powiadamiać inne obiekty, nie przyjmując żadnych założeń co do tego, co te obiekty reprezentują.

Programowanie zdarzeniowe



Oprogramowanie oparte na zdarzeniach – podsumowanie

- Metodyka szeroko wykorzystywana do tworzenia aplikacji z interfejsem graficznym.
- Często w systemie implementowana jest pętla obsługi zdarzeń, do której trafiają konkretne zdarzenia i która informuje inne moduły o zaistniałym zdarzeniu.
- Mechanizm sterujący pętlą komunikatów jest zazwyczaj niedostępny dla programisty. Sposób obsługi pętli jest ukryty przed programistą.
- Programista decyduje, które zdarzenia należy obsłużyć, w jaki sposób oraz jaki ma być efekt obsługi danego typu zdarzenia.
- Programista zazwyczaj nie decyduje o tym, jaki mechanizm jest wykorzystany do powiązania danego zdarzenia z procedurą jego obsługi – jest narzucony przez środowisko – pętla zdarzeń lub wzorzec obserwatora.

Programowanie zdarzeniowe

Zalety

- Duża elastyczność – procesory mogą być relatywnie proste, dedykowane wybranym zdarzeniom. Modyfikacja procesora lub dodanie nowego nie wymaga dużych zmian pozostałych elementów systemu.
- Elementy mogą być tworzone i testowane niezależnie i wdrażane stopniowo.
- Duża skalowalność – efektywna współpraca luźno powiązanych systemów (być może bardzo rozbudowanych).
- Duża wydajność – wykorzystanie wielu procesorów do równoległej obsługi danego zdarzenia (lub wielu zdarzeń danego rodzaju).
- Łatwość modyfikacji w obliczu zmieniających się wymagań – modyfikacji wymagają tylko wybrane procesory zdarzeń.
- Łatwość opisania interakcji z otoczeniem.
- Posługiwanie się inną metodologią podczas samego opisu funkcji obsługi zdarzeń (np. przetwarzanie strumieni danych w stylu funkcyjnym czy proceduralnym).

Wady

- Utrudnione uruchamianie, testowanie i wykrywanie błędów w systemie *jako całości* ze względu na asynchroniczny charakter zdarzeń (przetwarzanie równoległe).
- Konieczność uwzględnienia nieprawidłowego funkcjonowania poszczególnych komponentów.
- W przypadku korzystania z zewnętrznej biblioteki lub *frameworka*, np. *Qt*:
  - Skomplikowana budowa systemu – programowania w danym środowisku trzeba się nauczyć jak programowania w danym języku.
  - Konieczność znajomości możliwych zdarzeń i sposobu dostarczenia ich parametrów – mozolne przeglądanie dokumentacji.
  - Wykorzystanie ukrytej, lecz de facto istniejącej części programu odpowiadającej za obsługę pętli komunikatów – dodatkowe sprzężenia.