

Diagramy stanów – tworzenie modeli analizy i projektowania

Na podstawie

UML 2.0 Tutorial

http://sparxsystems.com.au/resources/uml2_tutorial/

Zofia Kruczkiewicz

Diagramy stanów – tworzenie modeli analizy i projektowania

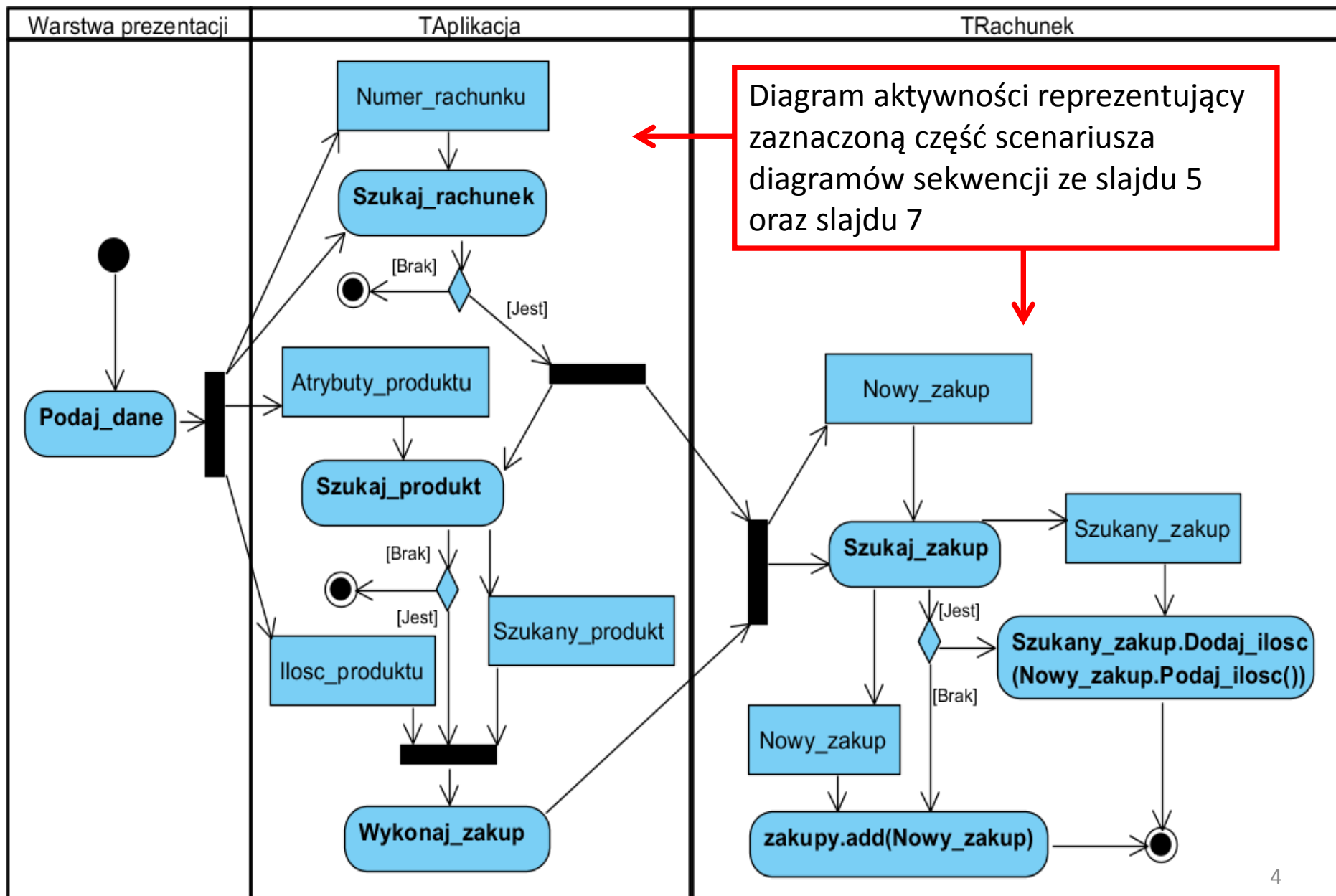
1. Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności - porównanie
2. Diagramy stanów UML

http://sparxsystems.com.au/resources/uml2_tutorial/

Diagramy stanów – tworzenie modeli analizy i projektowania

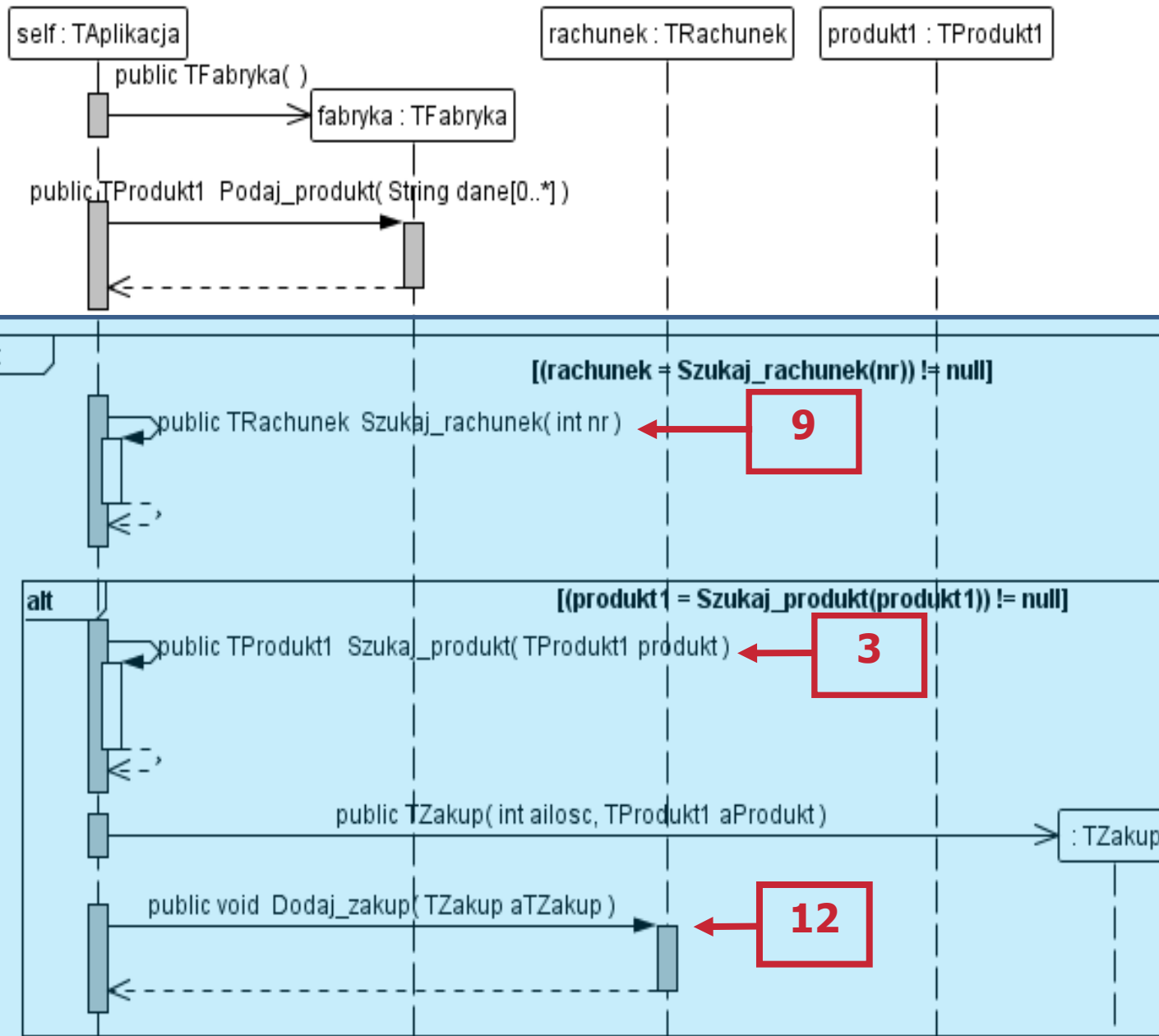
1. **Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności - porównanie**

Diagram czynności przypadku użycia *Wstawianie nowego zakupu* (model przypadku użycia w warstwie biznesowej)



(11) Wstawianie nowego zakupu

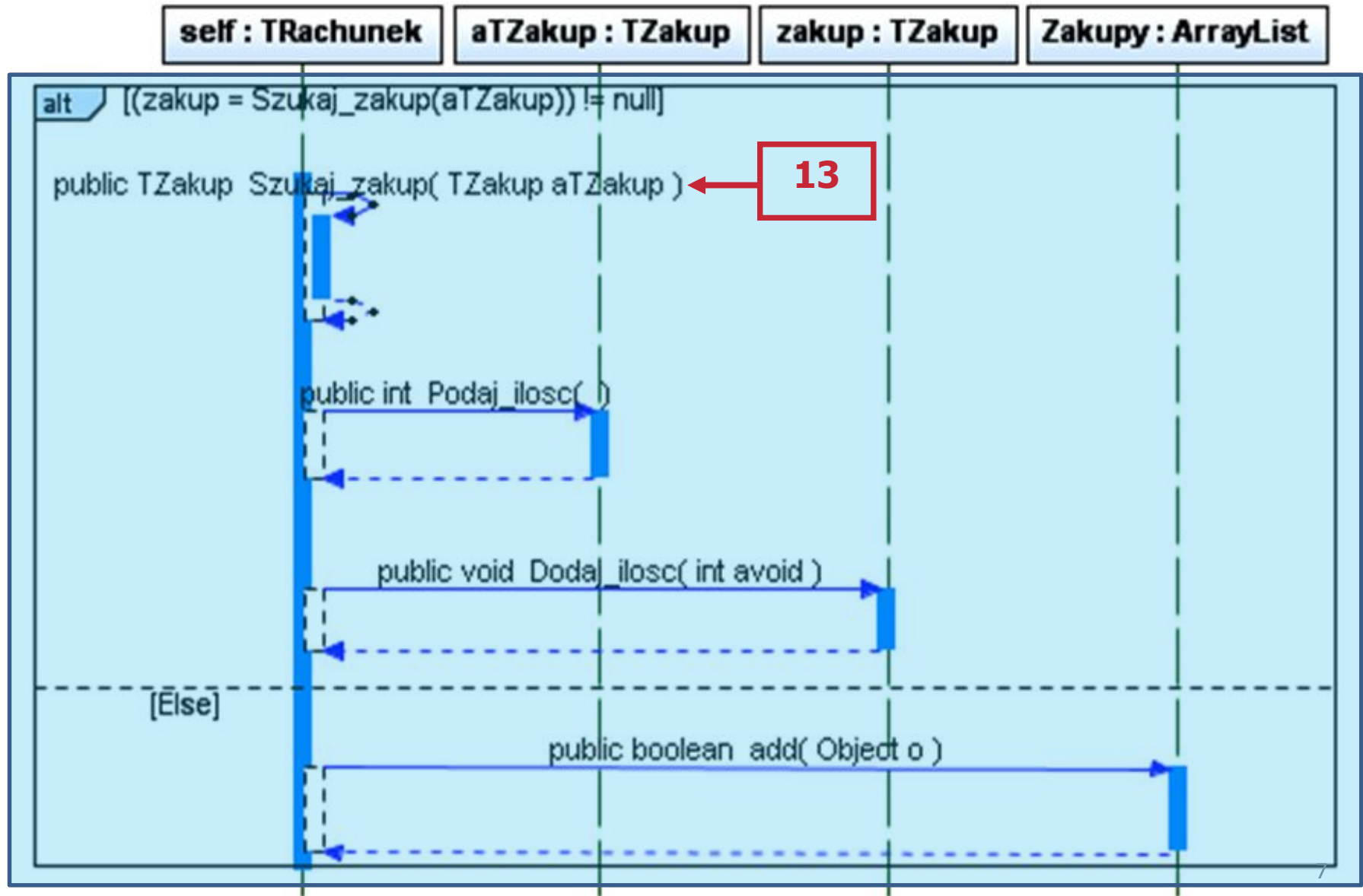
void TAplikacja::Wstaw_zakup (int nr, int ailosc, String dane[])



```
//TAplikacja
```

```
public void Wstaw_zakup (int nr, int ile, String dane[])  
{  
    TRachunek rachunek;  
    TFabryka fabryka = new TFabryka();  
    TProdukt1 produkt1 = fabryka.Podaj_produkt(dane);  
    if ((rachunek=Szukaj_rachunek(nr)) != null)  
        if ((produkt1=Szukaj_produkt(produkt1)) != null)  
            rachunek.Dodaj_zakup(new TZakup(ile, produkt1));  
}
```

(12) void TRachunek::Dodaj zakup(TZakup aTZakup)



```
//TRachunek
```

```
private ArrayList<TZakup> Zakupy =  
        new ArrayList<TZakup>();
```

```
public void Dodaj_zakup (TZakup aTZakup)  
{  
    TZakup zakup;  
    if ((zakup = Szukaj_zakup(aTZakup)) != null)  
        zakup.Dodaj_ilosc(aTZakup.Podaj_ilosc());  
    else  
        Zakupy.add(aTZakup);  
}
```


Diagramy stanów – tworzenie modeli analizy i projektowania

1. Modelowanie aktywności za pomocą diagramów sekwencji i aktywności - porównanie

2. **Diagramy stanów UML**

http://sparxsystems.com.au/resources/uml2_tutorial/

Diagramy stanów UML 2

Na podstawie

UML 2.0 Tutorial

http://sparxsystems.com.au/resources/uml2_tutorial/

Diagramy stanów – tworzenie modeli analizy i projektowania

1. Diagramy stanów UML

http://sparxsystems.com.au/resources/uml2_tutorial/

2. Przykład diagramów stanów UML – modelowanie wpływu przypadków użycia na stany obiektu

Diagramy stanów – tworzenie modeli analizy i projektowania

1. Diagramy stanów UML

http://sparxsystems.com.au/resources/uml2_tutorial/

Dwa rodzaje diagramów UML 2

Diagramy UML modelowania strukturalnego

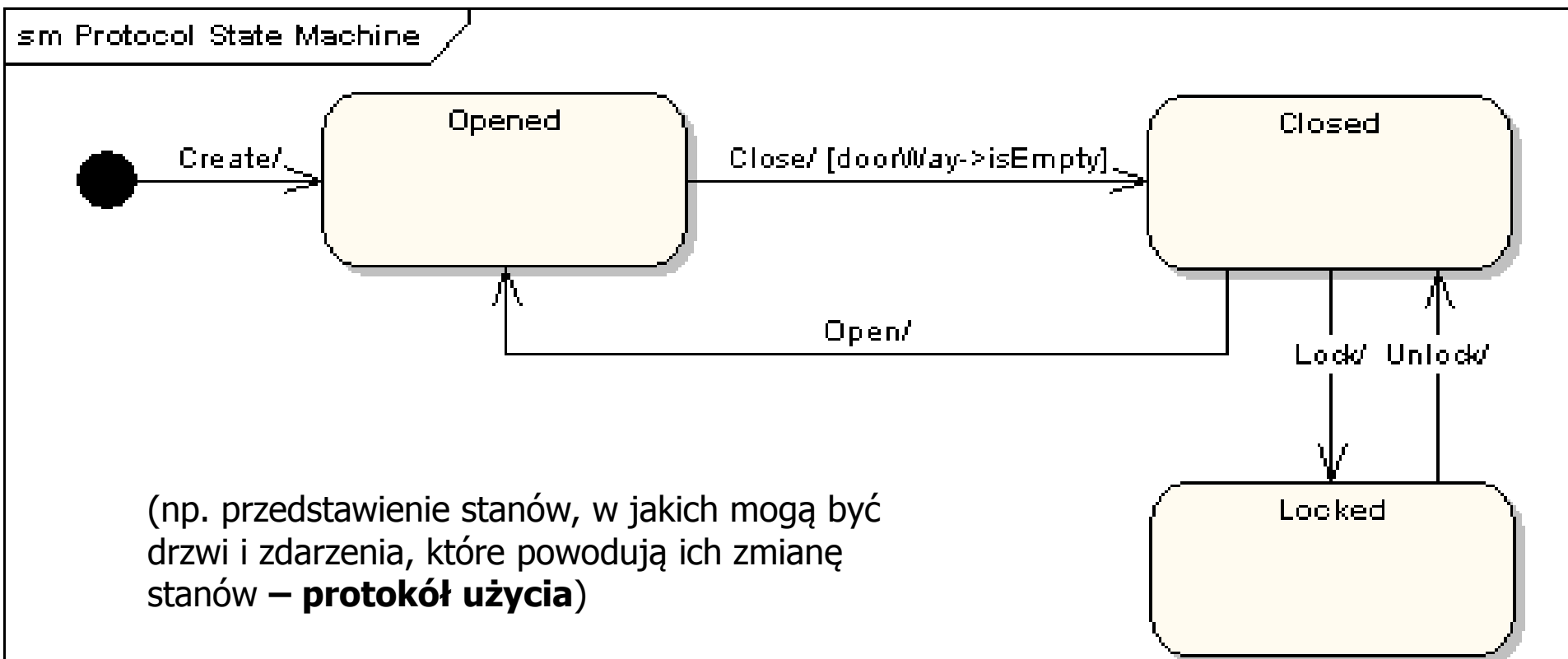
- Diagramy pakietów
- *Diagramy klas*
- Diagramy obiektów
- Diagramy mieszane
- Diagramy komponentów
- Diagramy wdrożenia

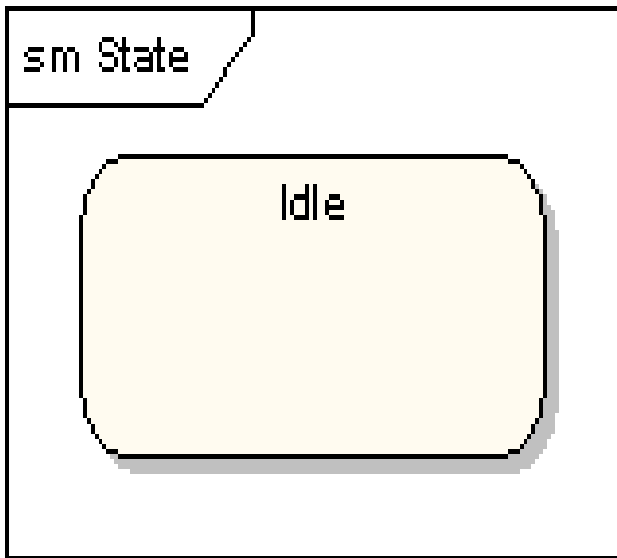
Diagramy UML modelowania zachowania

- *Diagramy przypadków użycia*
- *Diagramy aktywności*
- *Diagramy stanów*
- Diagramy komunikacji
- *Diagramy sekwencji*
- Diagramy czasu
- Diagramy interakcji

Diagram stanu opisuje zmiany stanu obiektu, podsystemu lub systemu pod wpływem działania operacji - jest szczególnie przydatny, gdy zachowanie obiektu jest złożone. Przedstawia on **maszynę stanów** jako przepływ sterowania między stanami.

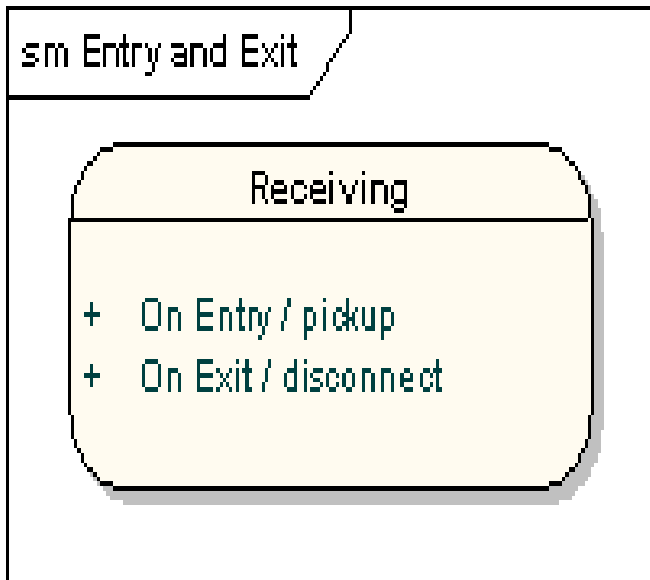
Diagram stanów jest grafem złożonym z wierzchołków i krawędzi: wierzchołkami są **stany** (prostokąty o zaokrąglonych rogach), krawędziami są **przejścia** (strzałki).





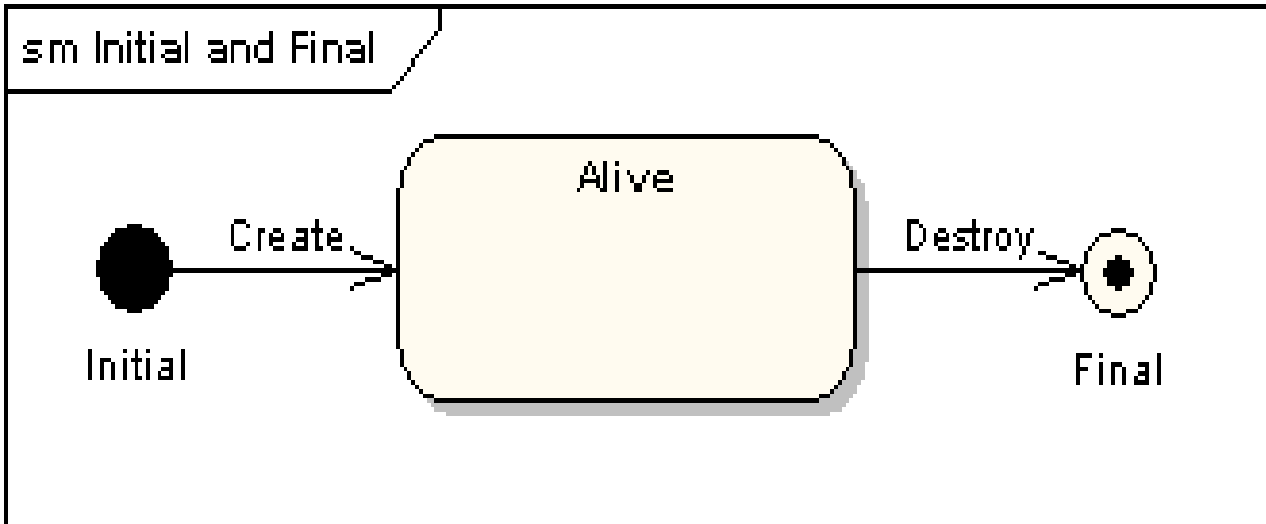
Stan jest okolicznością lub sytuacją, w jakiej znajduje się obiekt

- jest rezultatem poprzedniej aktywności
- spełnia jakiś warunek
- jest określony przez wartości własnych atrybutów i powiązań do innych zadań
- wykonuje pewne czynności
- czeka na jakieś zdarzenie



• **Nazwa** - unikatowy ciąg znaków, brak nazwy dla stanu anonimowego

• **Akcje wejściowe (entry)** i **wyjściowe (exit)**
- akcje wykonywane odpowiednio przy wejściu do stanu i przy wyjściu)

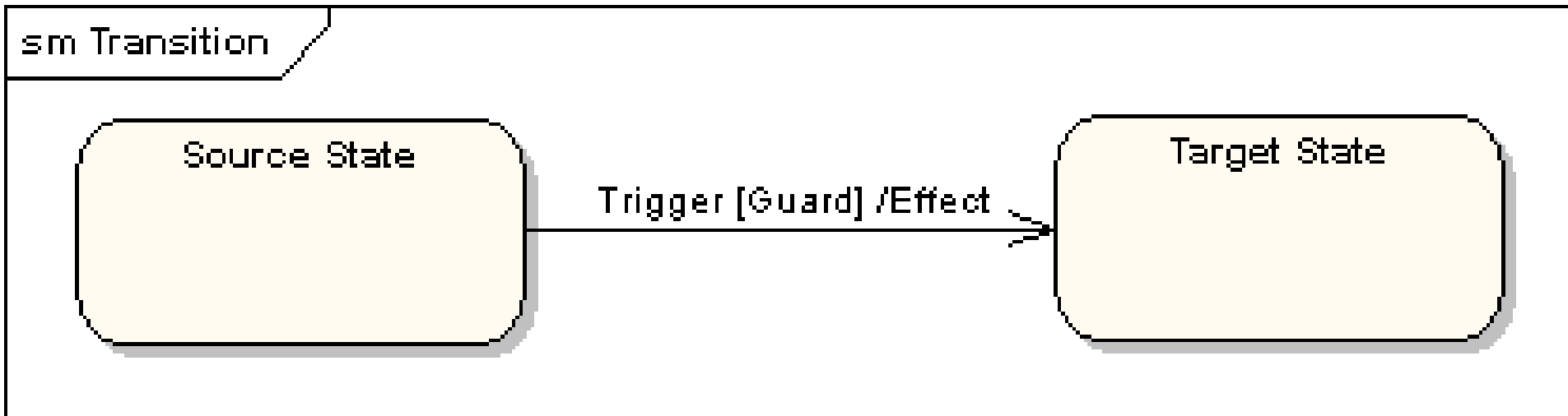


Stan początkowy

(Initial) – może być tylko jeden stan początkowy

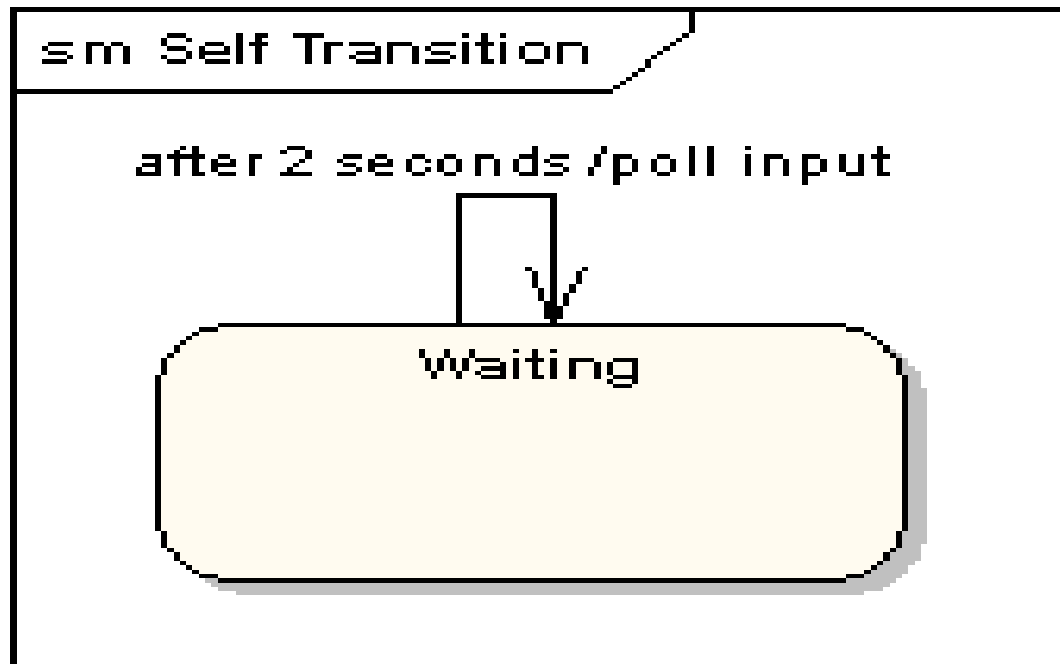
Stan końcowy (Final) –

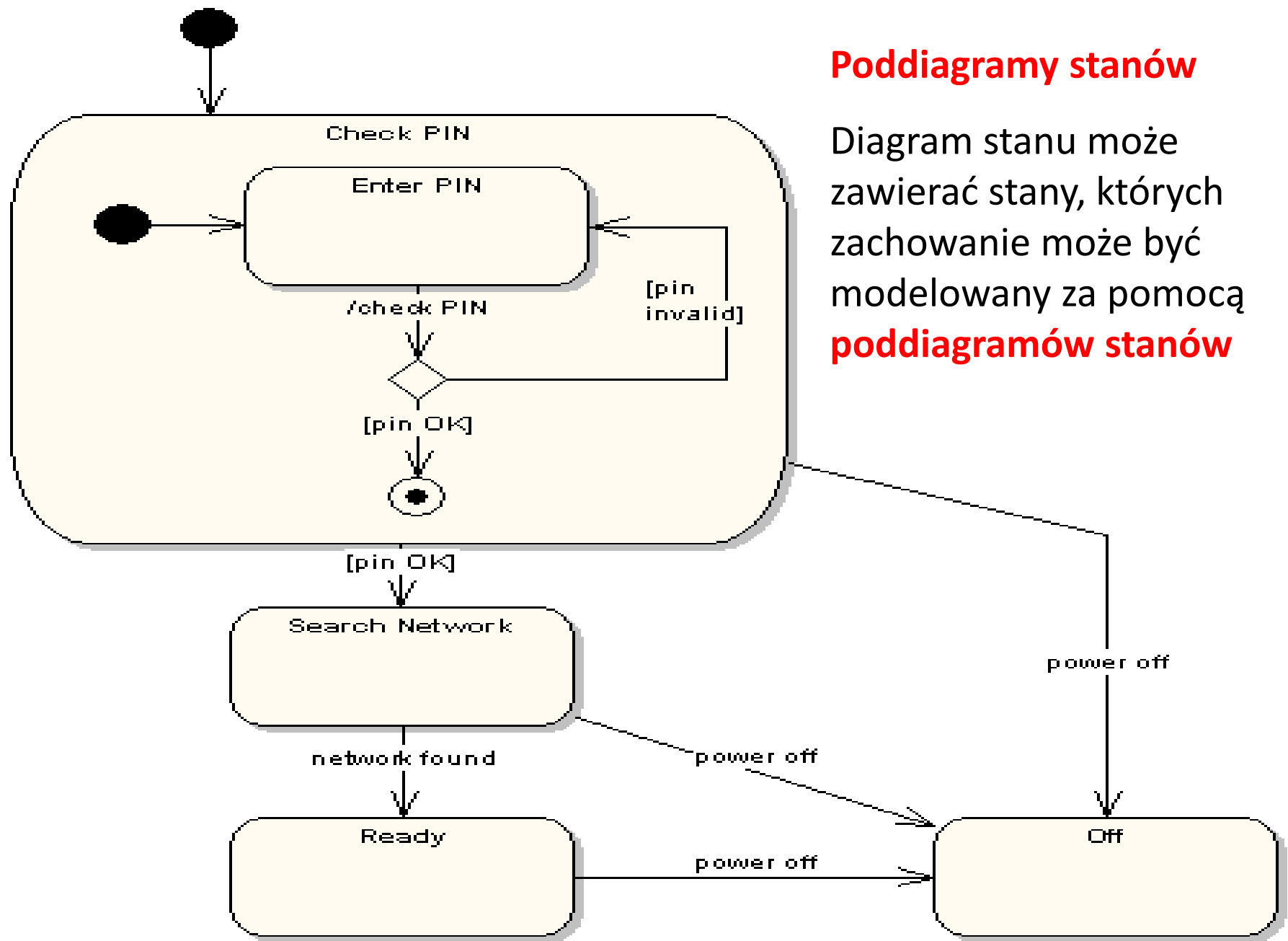
może być kilka stanów końcowych



Przejście (Transition) jest związkiem między dwoma stanami, który wskazuje, że np. obiekt znajdujący się w pierwszym stanie wykona pewne **akcje (Effect)** i przejdzie do drugiego stanu, ilekroć zaistnieje określone **zdarzenie (Trigger)** i będą spełnione określone **warunki (Guard)**.

Przejście własne jest związkiem między tym samym stanem, który wskazuje, że np. obiekt znajdujący się w pewnym stanie wykona pewne **akcje** i powróci do tego samego stanu, ilekroć zaistnieje określone **zdarzenie** i będą spełnione określone **warunki**.

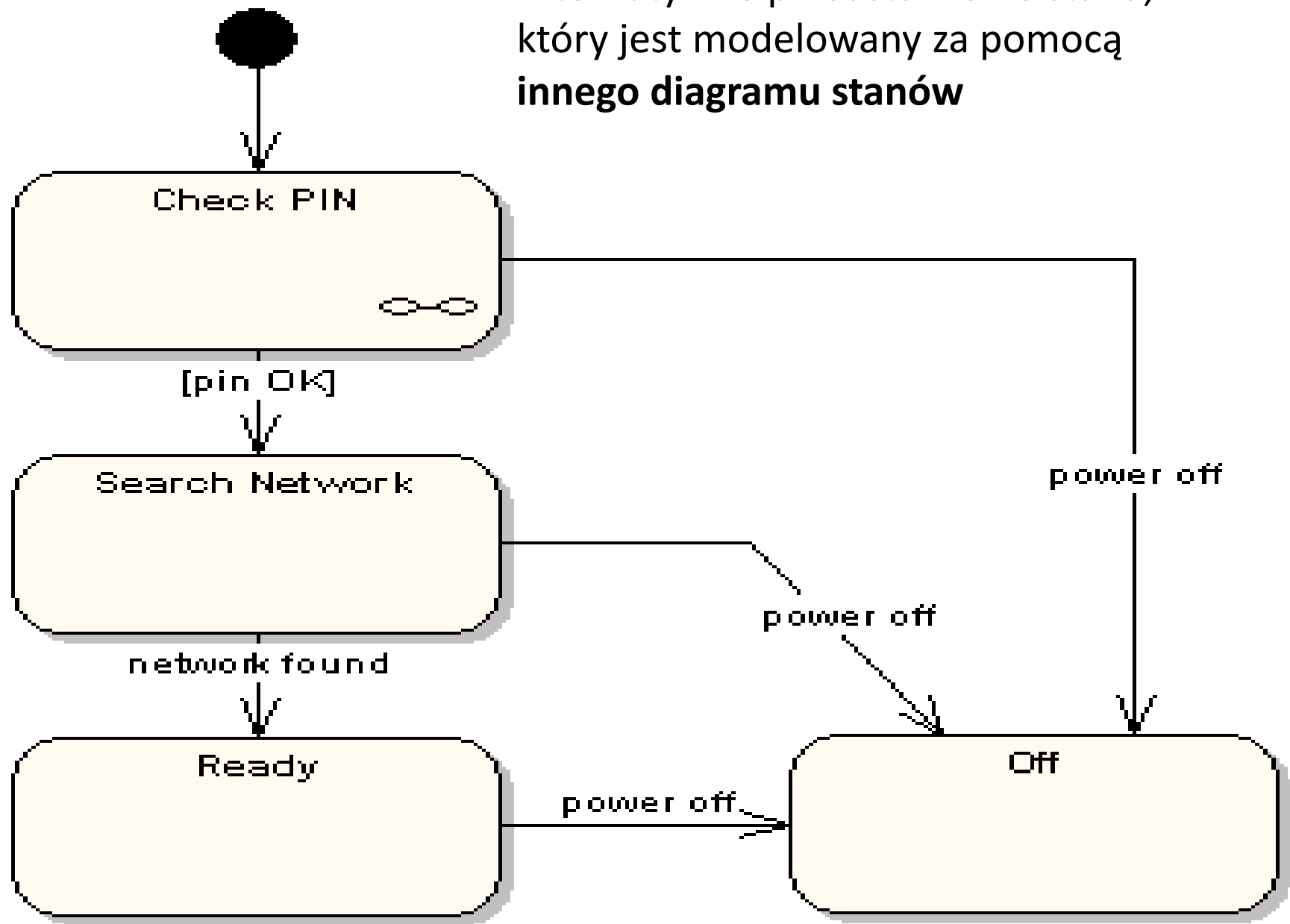


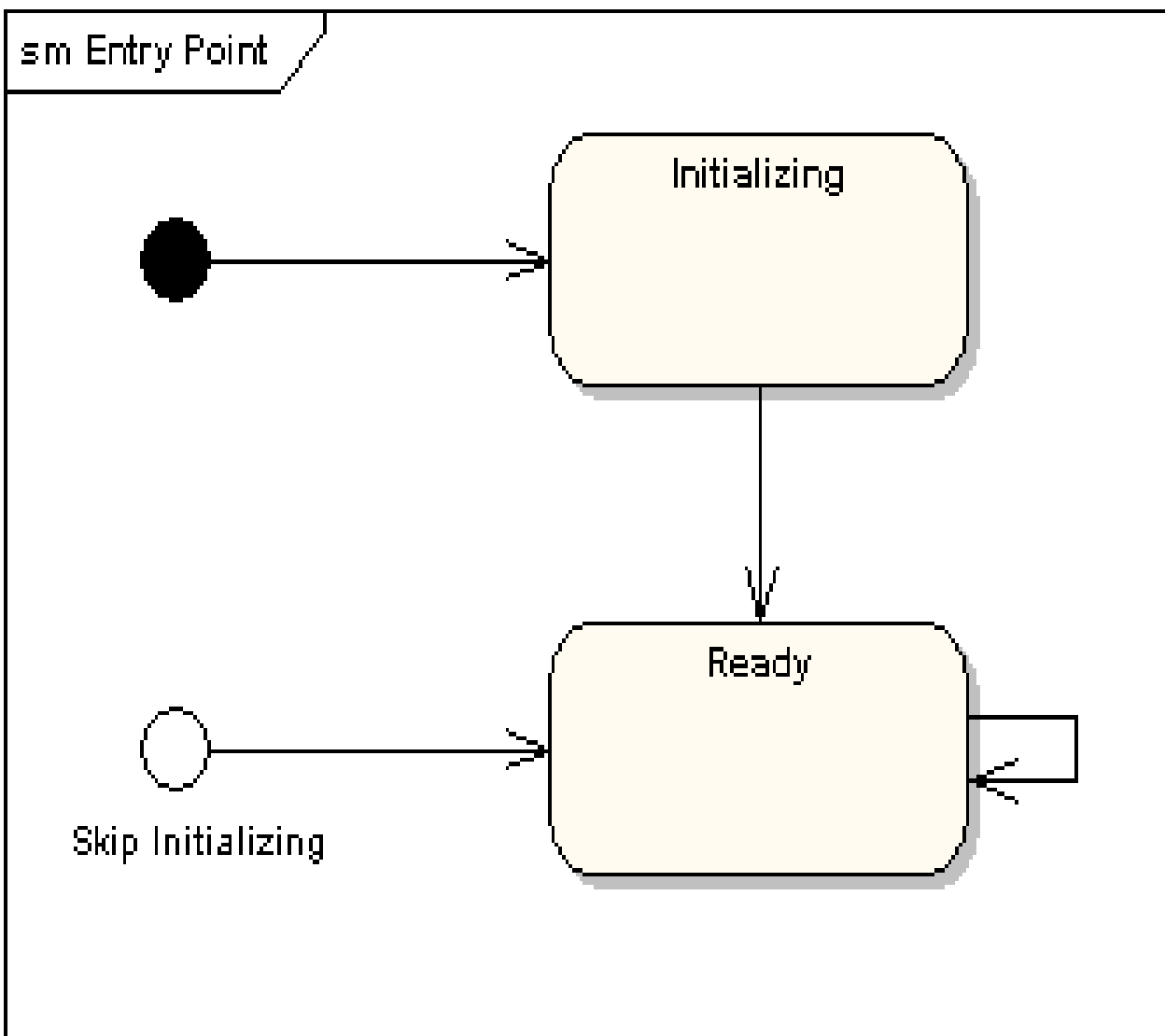


Poddiagramy stanów

Diagram stanu może zawierać stany, których zachowanie może być modelowane za pomocą **poddiagramów stanów**

Alternatywne przedstawienie stanu, który jest modelowany za pomocą innego diagramu stanów





Stany początkowe w poddiagramach stanów

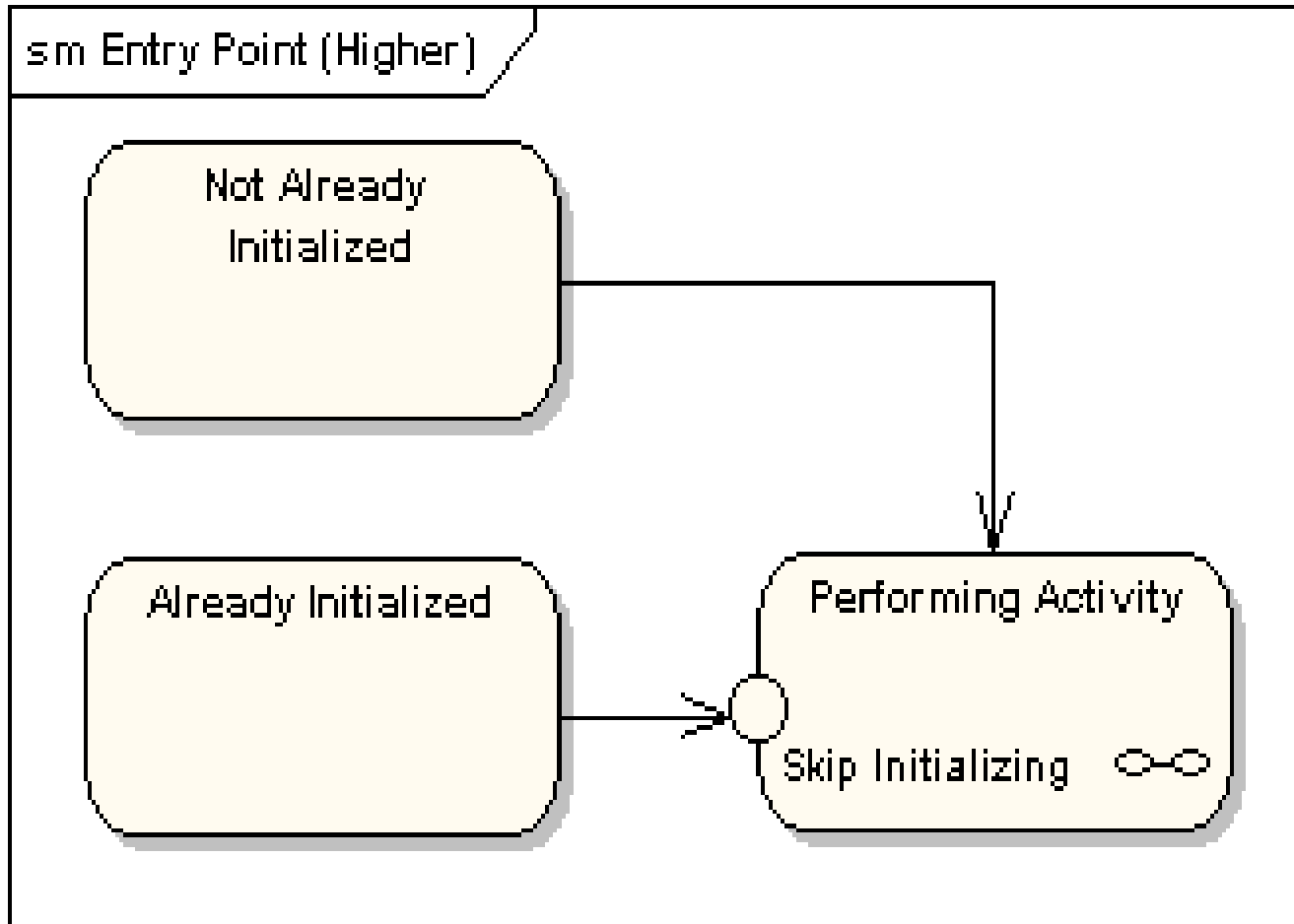
Wskazanie różnych stanów początkowych w poddiagramie stanów:

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

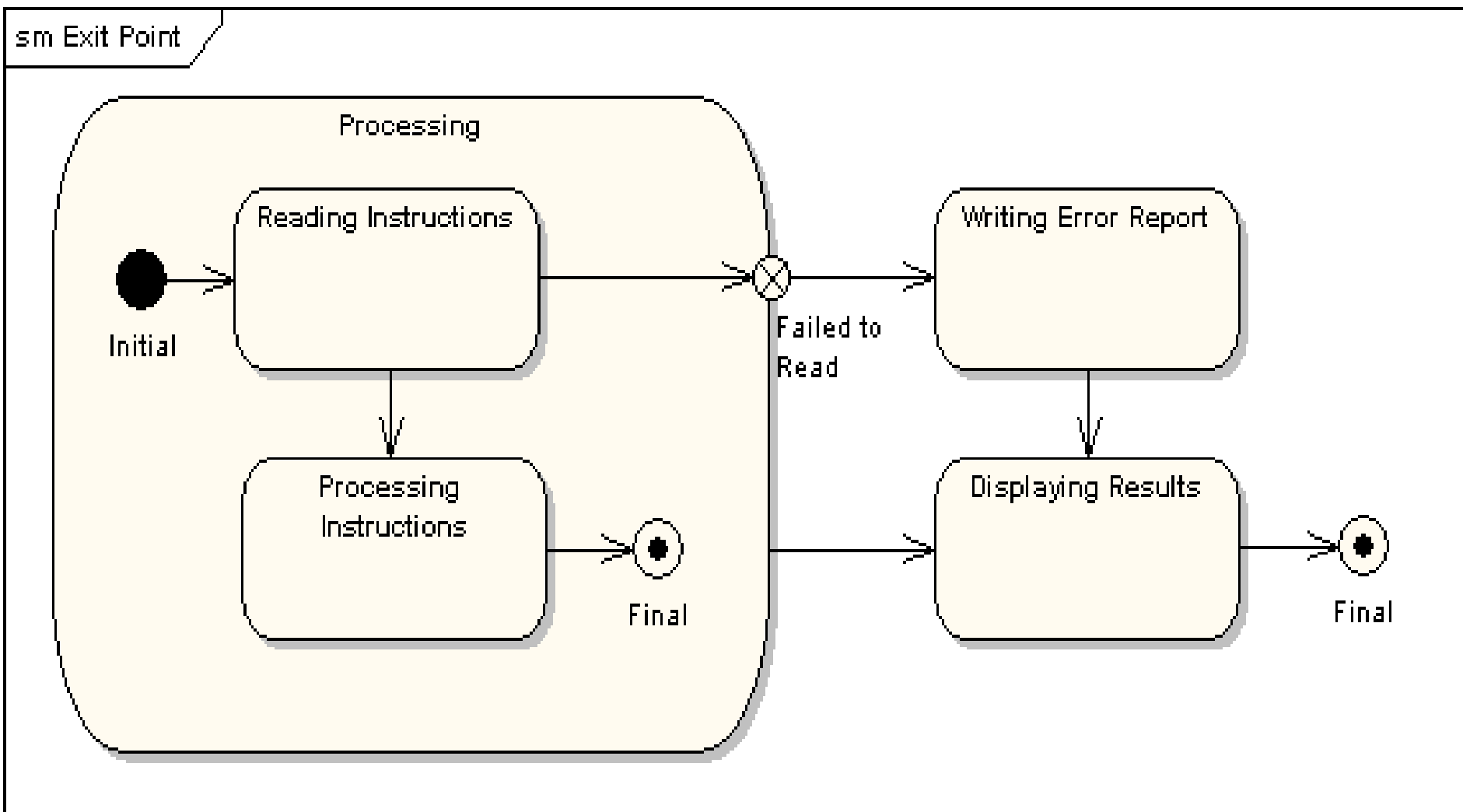
Punkty startowe w diagramach nadrzędnych

Diagram stanów zawierający różne punkty startowe dla poddiagramów stanów (reprezentowanych przez inne diagramy):

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

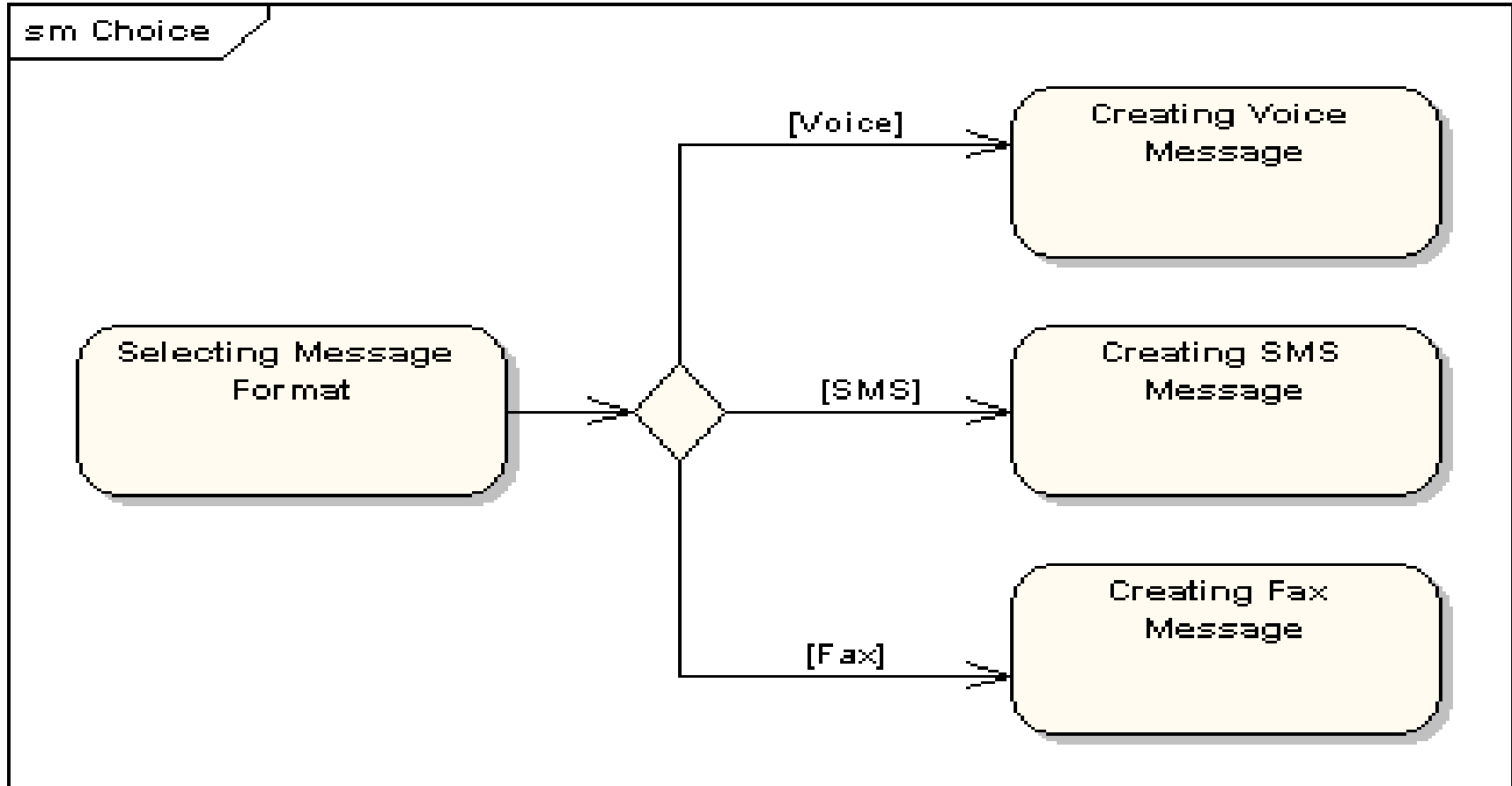


Punkt wyjścia – modelowanie osiągnięcia alternatywnych stanów końcowych (**Final**) przez obiekt

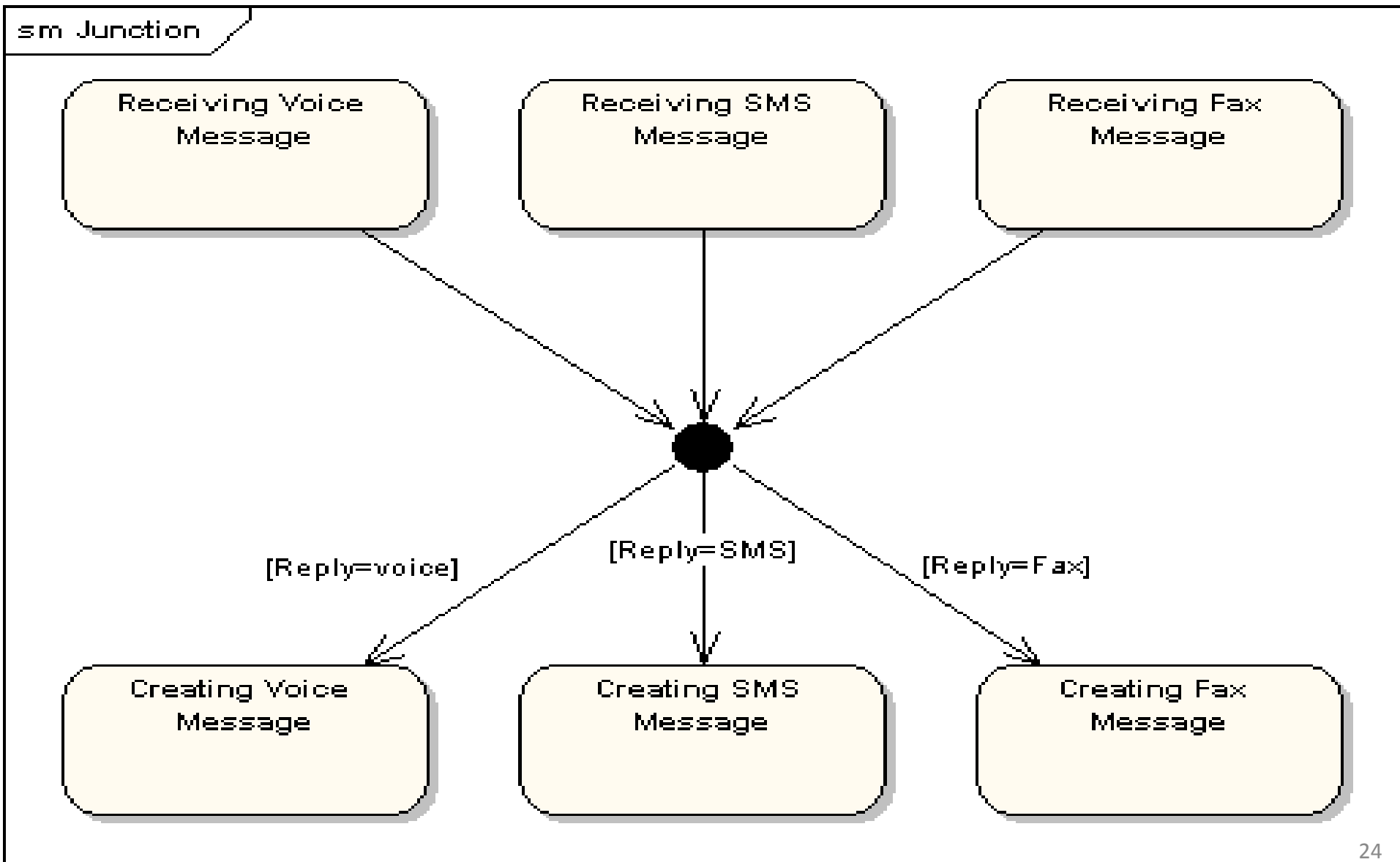


Pseudo stan wyboru:

- jedno przejście ze stanu wejściowego do **pseudo stanu wyboru (romb)** i kilka przejść na wyjściu tego pseudo stanu
- w wyniku zdarzenia następuje przejście ze stanu wejściowego (np. Selecting Message Format) i na podstawie spełnionego warunku wybór przejścia do jednego ze stanów wyjściowych (np. wybór przejścia na podstawie wybranego formatu wiadomości w stanie wejściowym); dynamiczny charakter wyboru przejścia

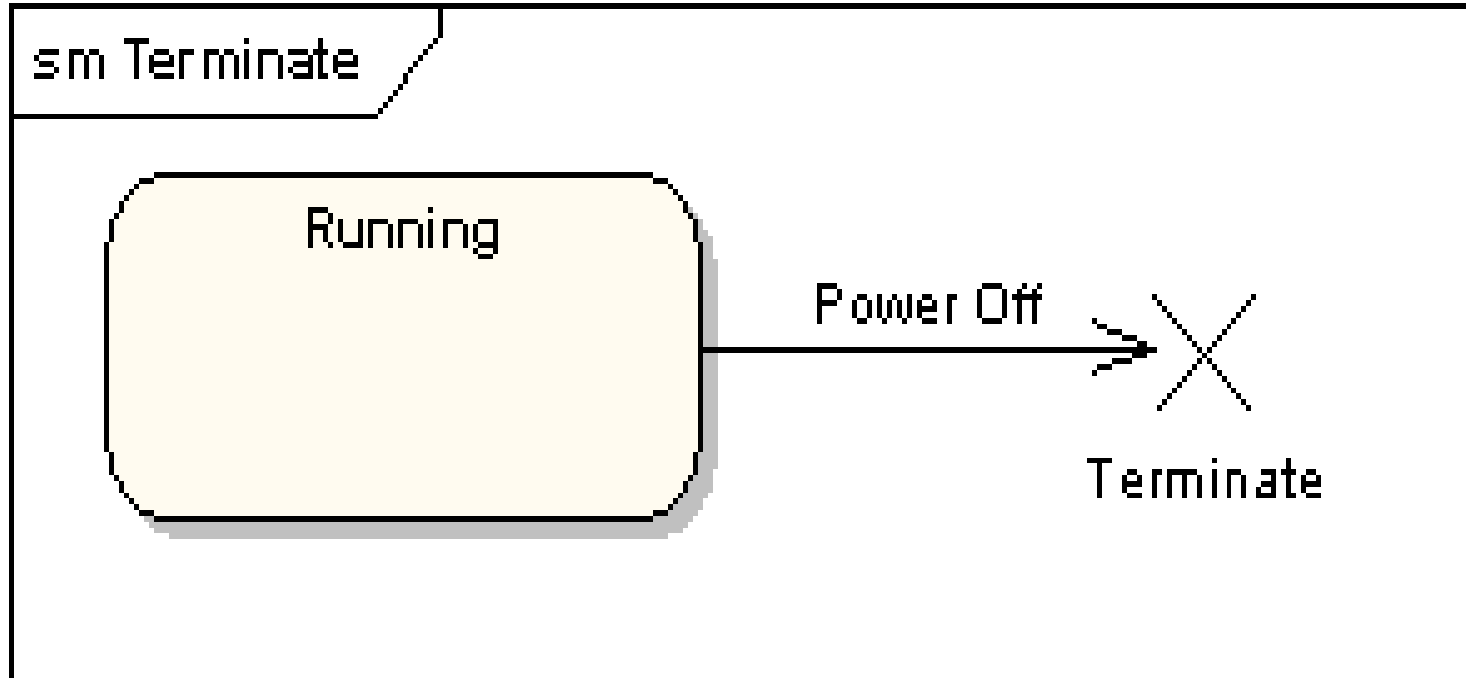


Pseudo stan typu połączenie – w pseudo stanie typu połączenie możliwość wyboru przejść do stanów wyjściowych po zdarzeniach zachodzących na przejściach ze stanów wejściowych

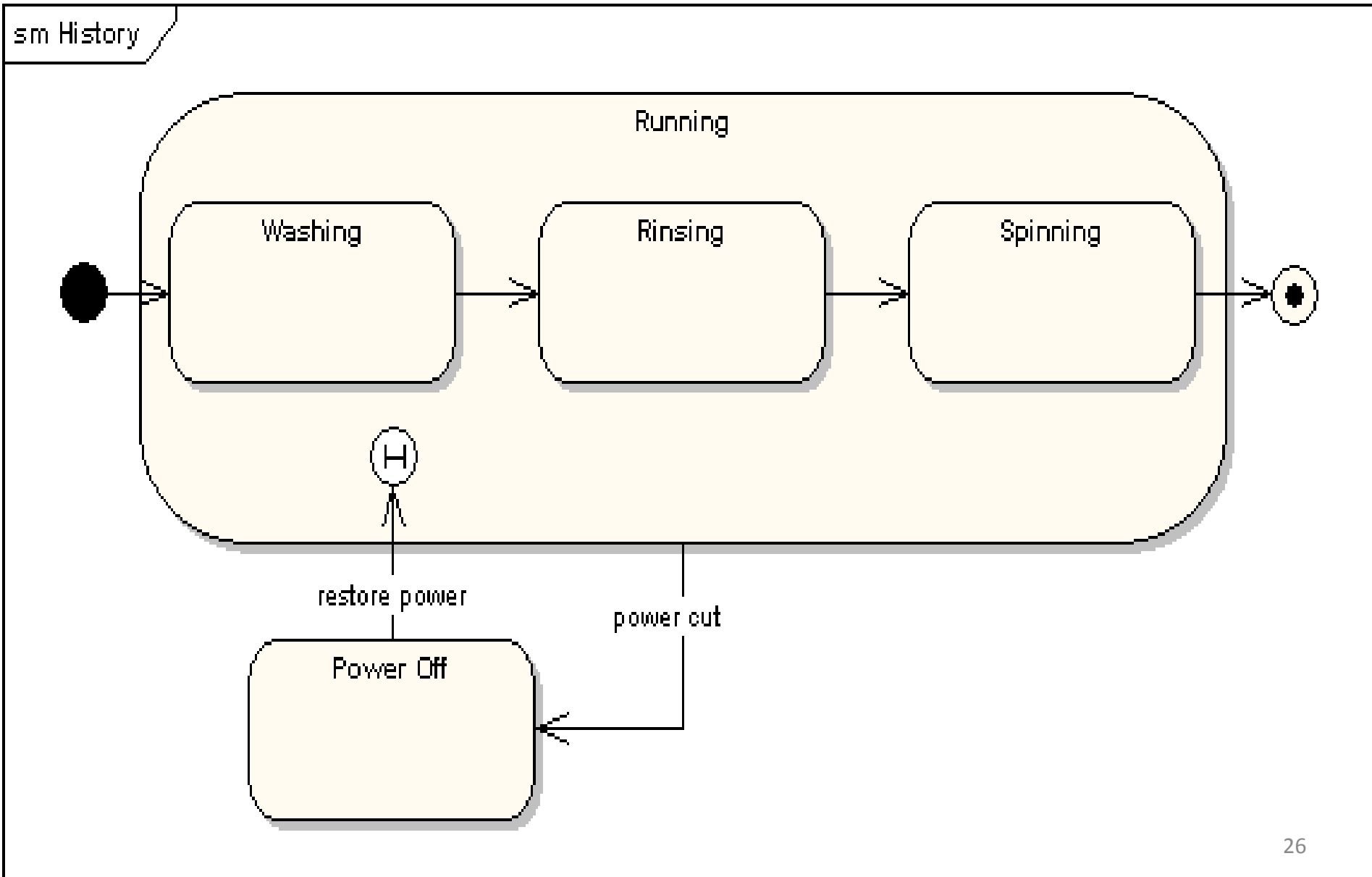


Pseudo stan typu zakończenie

oznacza zakończenie linii życia maszyny stanowej

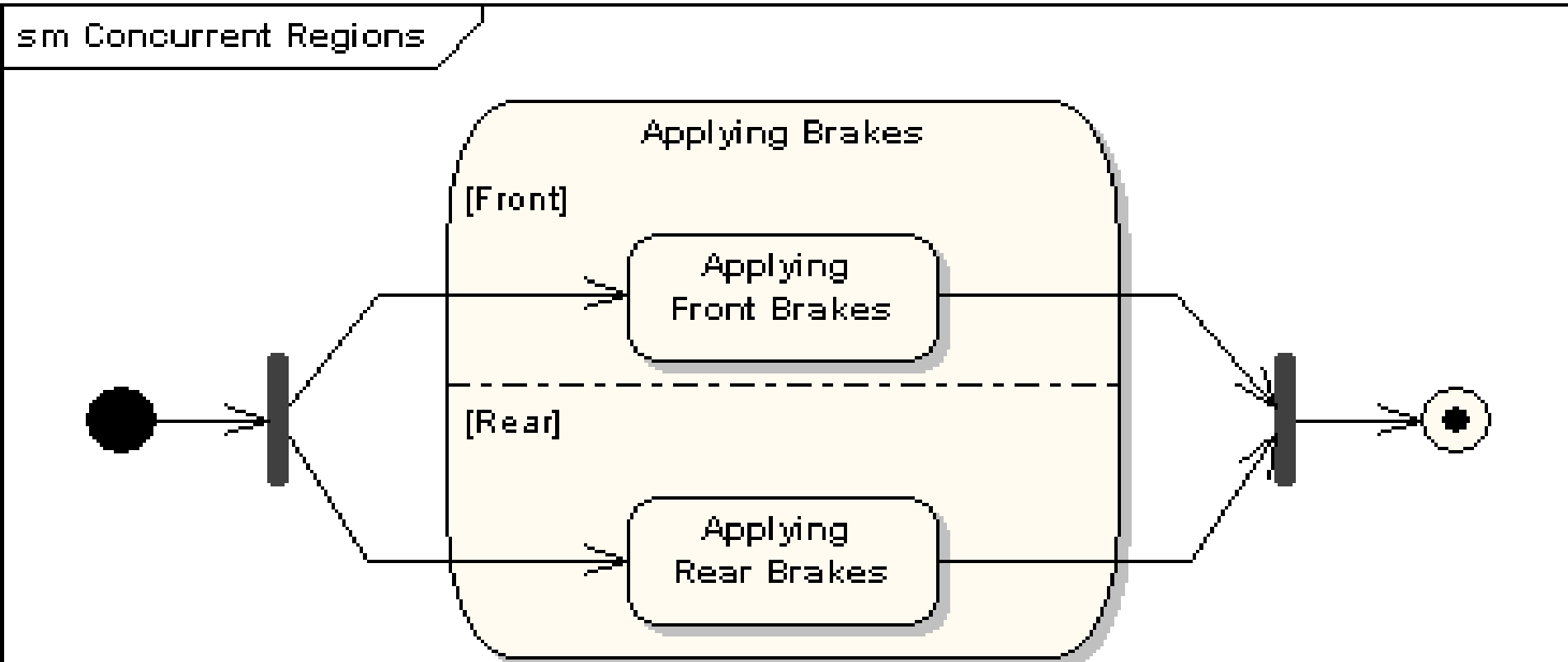


Stany historyczne – przedstawiają stany wcześniejsze (historyczne) przed przerwaniem działania maszyny stanowej (np. w chwili załączenia zasilania maszyna stanowa zmywarki pamięta stan, w którym ma wznowić działanie)



Równoległe podstany

Stan może być podzielony między równoległe podstany wykonywane jednocześnie. (np. sterowanie przednimi (front) i tylnymi (rear) hamulcami odbywa się równoległe i musi być zsynchronizowane – wyrażone za pomocą symbolu rozdzielenia na pseudo stany oraz symbolu połączenia pseudo stanów. Równoległe podstany są używane do modelowania synchronizacji wątków



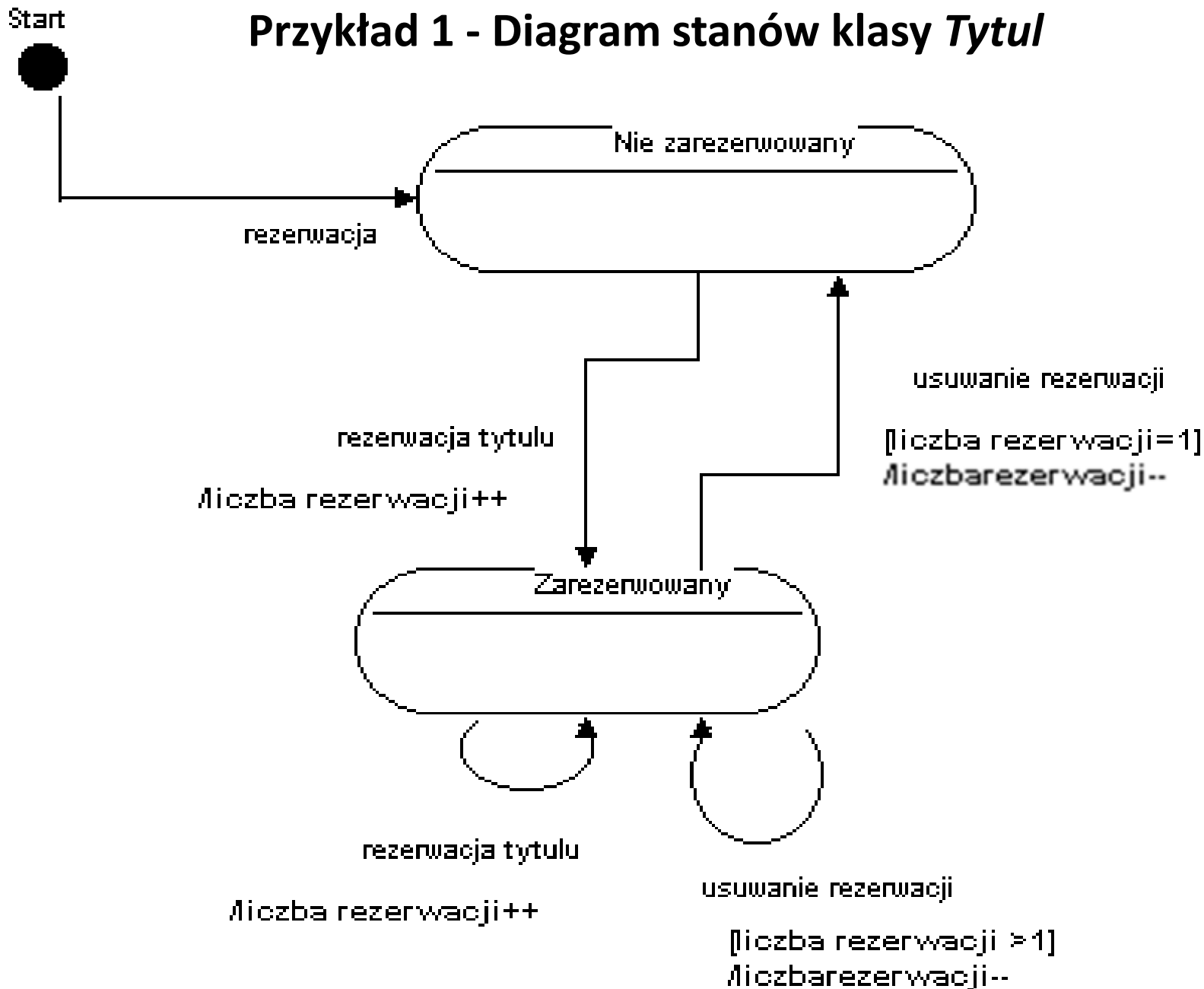
Diagramy stanów – tworzenie modeli analizy i projektowania

1. Diagramy stanów UML

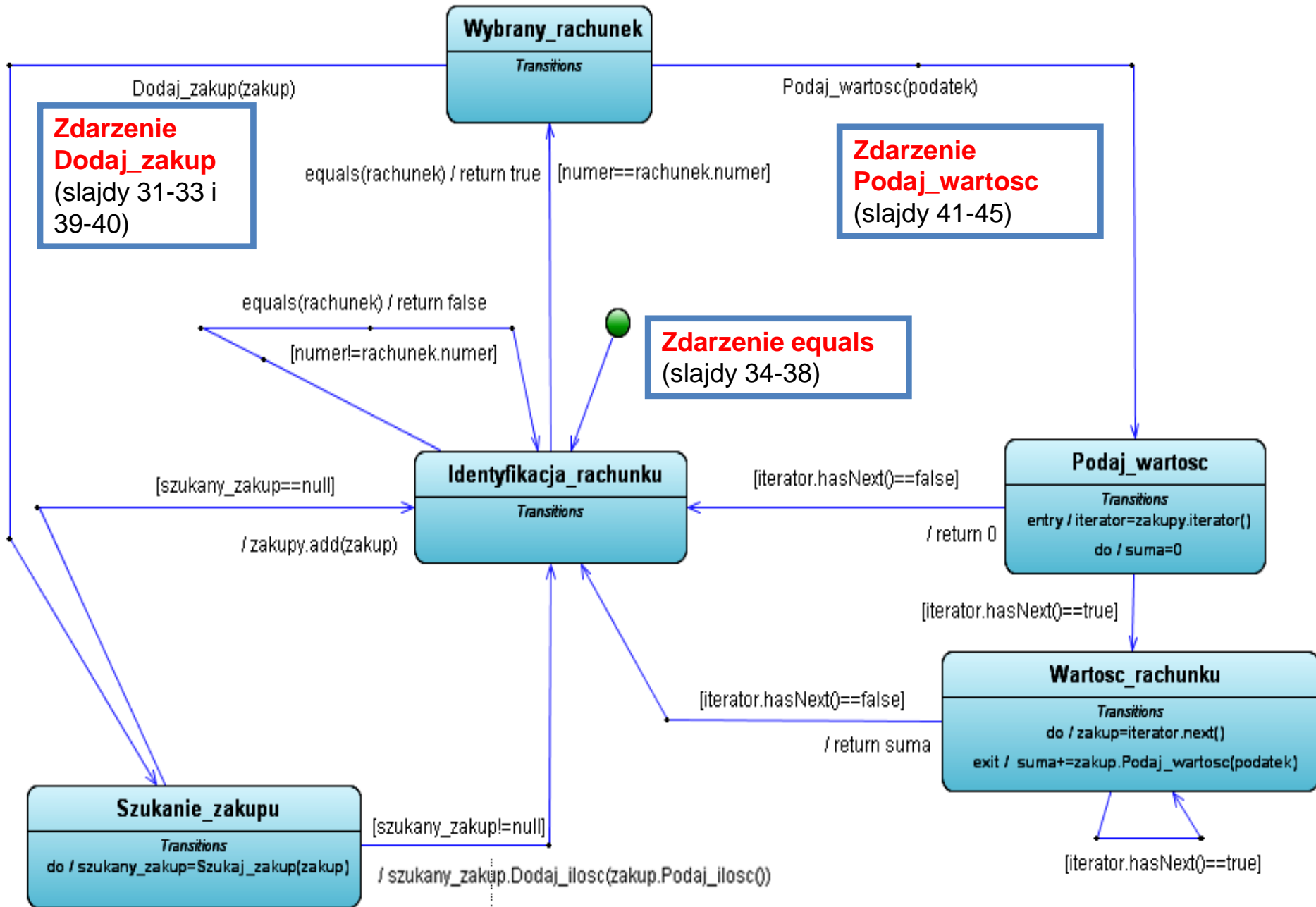
http://sparxsystems.com.au/resources/uml2_tutorial/

2. Przykład diagramów stanów UML – modelowanie wpływu przypadków użycia na stany obiektu

Przykład 1 - Diagram stanów klasy *Tytul*



Przykład 2 - Klasa *TRachunek*



Projekt przypadku użycia –
zdarzenie **Dodaj_zakup**

„**Wstawianie nowego zakupu**”

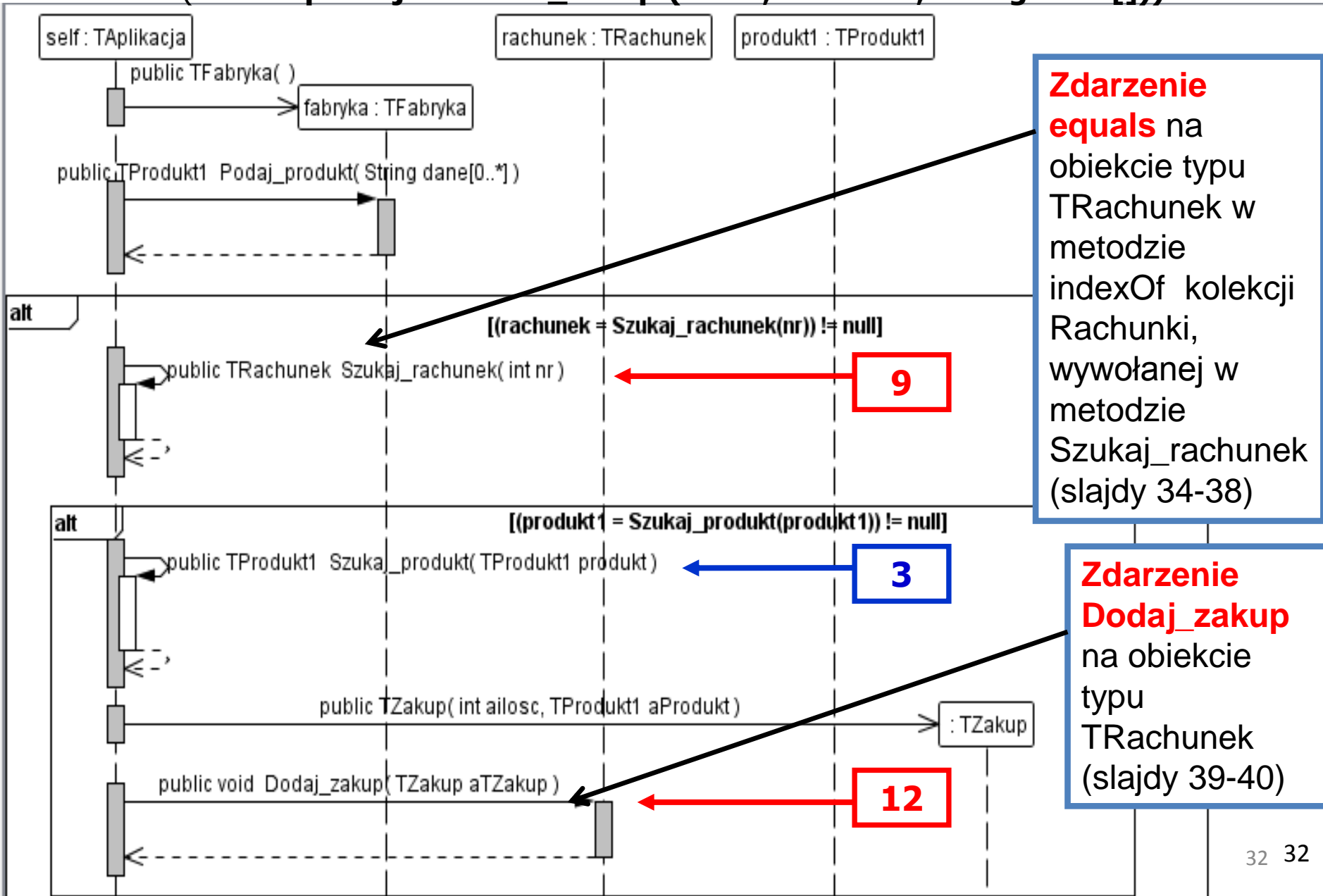
za pomocą diagramu sekwencji i diagramu klas.

Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

Definiowanie kodu metod realizujących przypadek użycia

na podstawie diagramów sekwencji

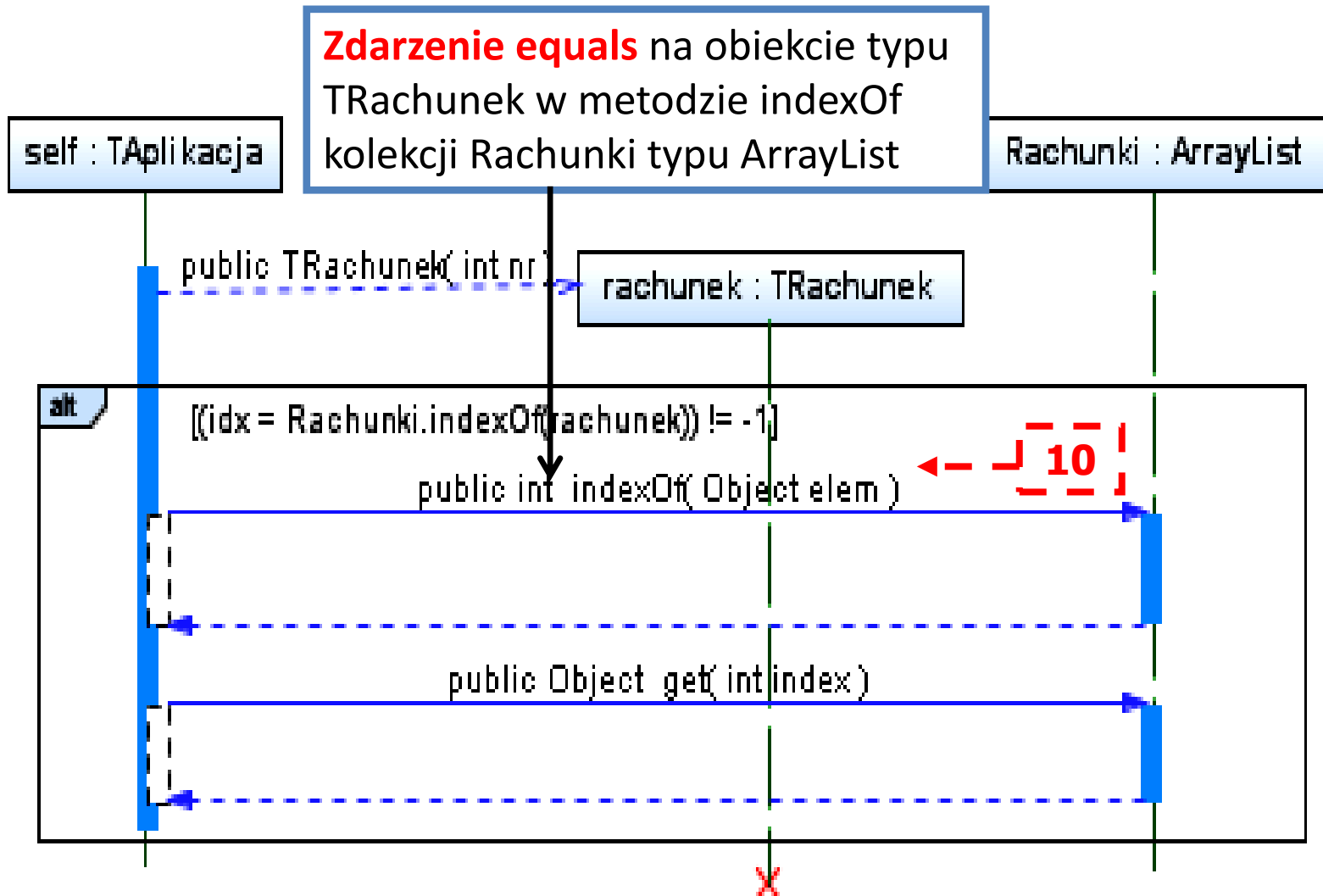
(11) Wstawianie nowego zakupu – generowanie zdarzeń equals i Dodaj_zakup na obiekcie typu TRachunek
(void TAplikacja::Wstaw_zakup (int nr, int ailosc, String dane[]))




```
public void Wstaw_zakup (int nr, int ile, String dane[])  
{  
    TRachunek rachunek;  
    TFabryka fabryka = new TFabryka();  
    TProdukt1 produkt1 = fabryka.Podaj_produkt(dane);  
    if ((rachunek=Szukaj_rachunek(nr)) != null)  
        if ((produkt1=Szukaj_produkt(produkt1)) != null)  
            rachunek.Dodaj_zakup(new TZakup(ile, produkt1));  
}
```

(9) Szukanie rachunku

(TRachunek TAplikacja::Szukaj_rachunek(int nr))



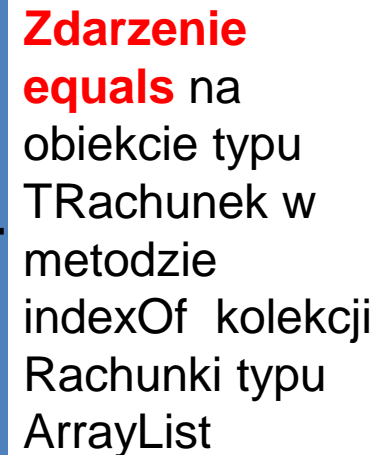
```
static private ArrayList <TRachunek> Rachunki =  
                new ArrayList <TRachunek>();
```

```
public TRachunek Szukaj_rachunek (int nr)  
{  
    TRachunek rachunek = new TRachunek(nr);  
    int idx;  
    if ((idx=Rachunki.indexOf(rachunek)) != -1)  
    {  
        rachunek=Rachunki.get(idx);  
        return rachunek;  
    }  
    return null;  
}
```

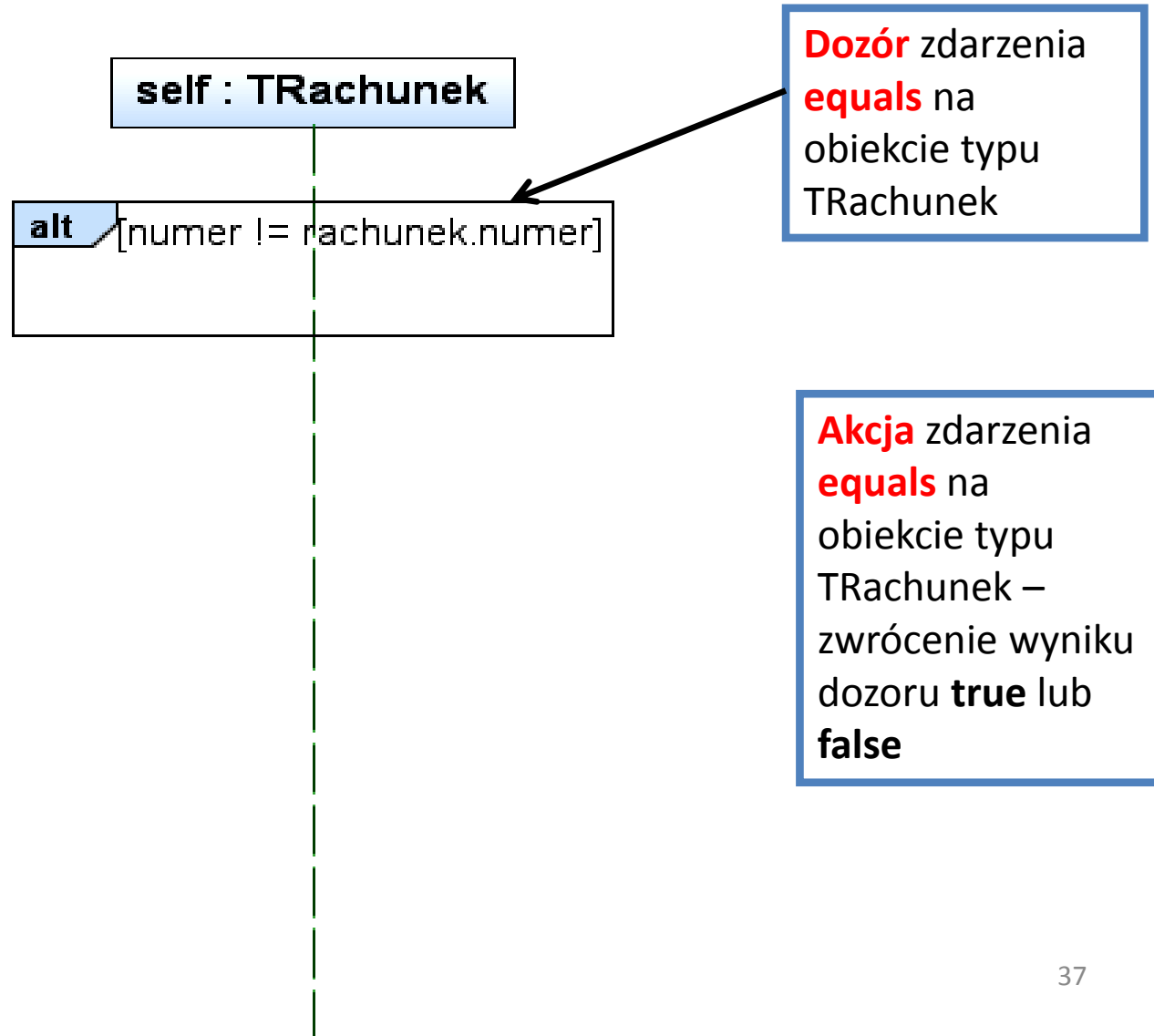
//metoda indexOf obiektu Rachunki klasy typu ArrayList

```
public int indexOf(Object o) {  
    if (o == null) {  
        for (int i = 0; i < size; i++)  
            if (elementData[i]==null)  
                return i;  
    } else {  
        for (int i = 0; i < size; i++)  
            if (o.equals(elementData[i]))  
                return i;  
    }  
    return -1;  
}
```

Zdarzenie equals na obiekcie typu TRachunek w metodzie indexOf kolekcji Rachunki typu ArrayList



(10) boolean TRachunek::equals(Object rachunek) - metoda zdarzeniowa



//TRachunek

//metoda zdarzeniowa equals

// metody użyte w kodzie metody są akcjami zdarzenia

//instrukcje warunkowe mogą być użyte jako dozory

public boolean equals (Object aTRachunek)

{

TRachunek rachunek= (TRachunek)aTRachunek;

boolean bStatus = **true**;

if (numer!= rachunek.numer)

bStatus = **false**;

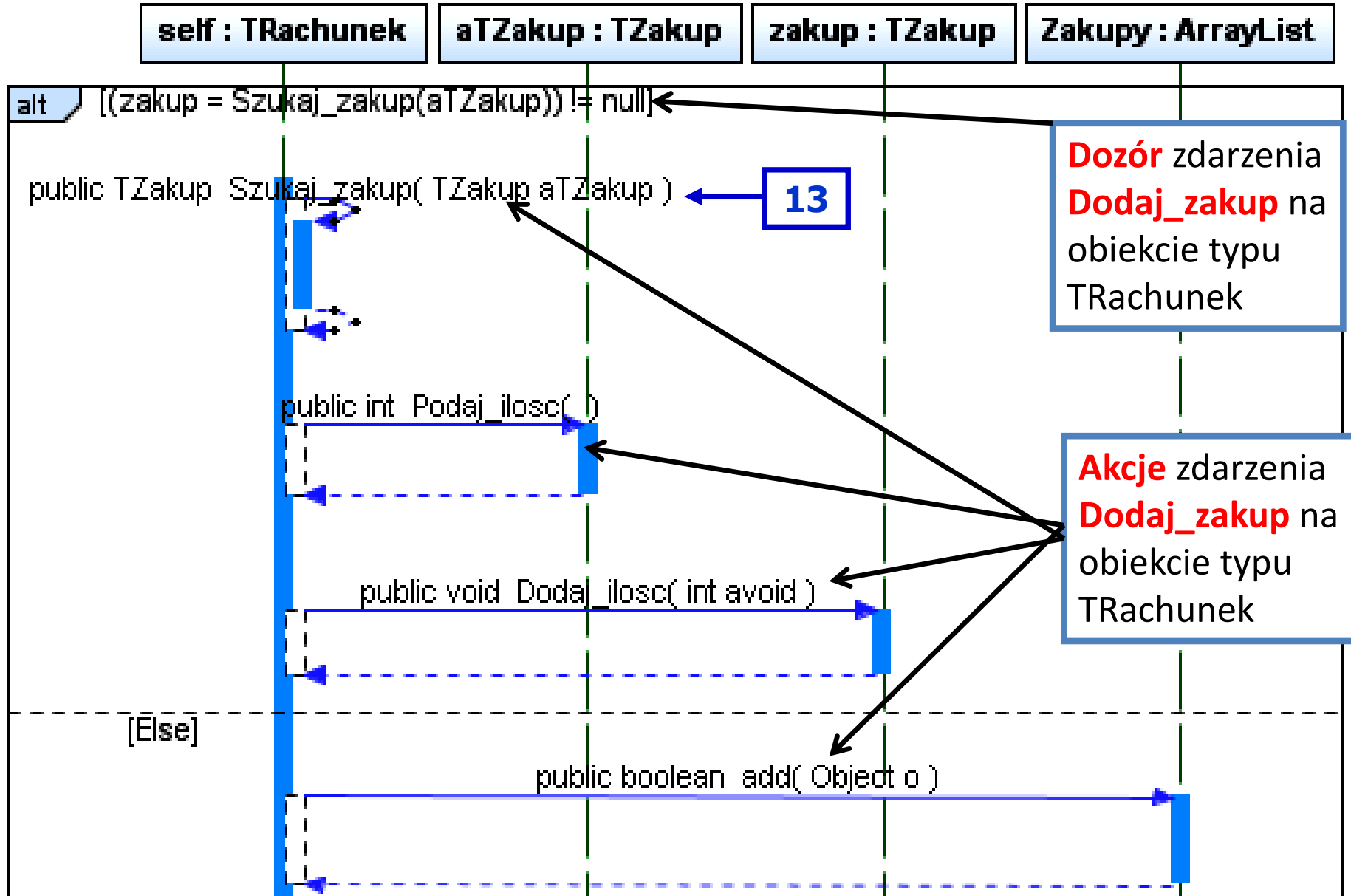
return bStatus;

}

Dozór zdarzenia **equals**
na obiekcie typu
TRachunek

Akcja zdarzenia **equals** na
obiekcie typu TRachunek –
zwrócenie wyniku dozoru
true lub false

(12) void TRachunek::Dodaj zakup(TZakup aTZakup) – metoda zdarzeniowa



```
//metoda zdarzeniowa Dodaj_zakup  
// metody użyte w kodzie metody są akcjami zdarzenia  
//instrukcje warunkowe mogą być użyte jako dozory
```

```
private ArrayList<TZakup> Zakupy =  
                new ArrayList<TZakup>();
```

```
public void Dodaj_zakup (TZakup aTZakup)  
{  
    TZakup zakup;  
    if ((zakup = Szukaj_zakup(aTZakup)) != null)  
        zakup.Dodaj_ilosc(aTZakup.Podaj_ilosc());  
    else  
        Zakupy.add(aTZakup);  
}
```


Projekt przypadku użycia
zdarzenie **Podaj_wartosc**

„**Obliczanie wartości rachunku**”

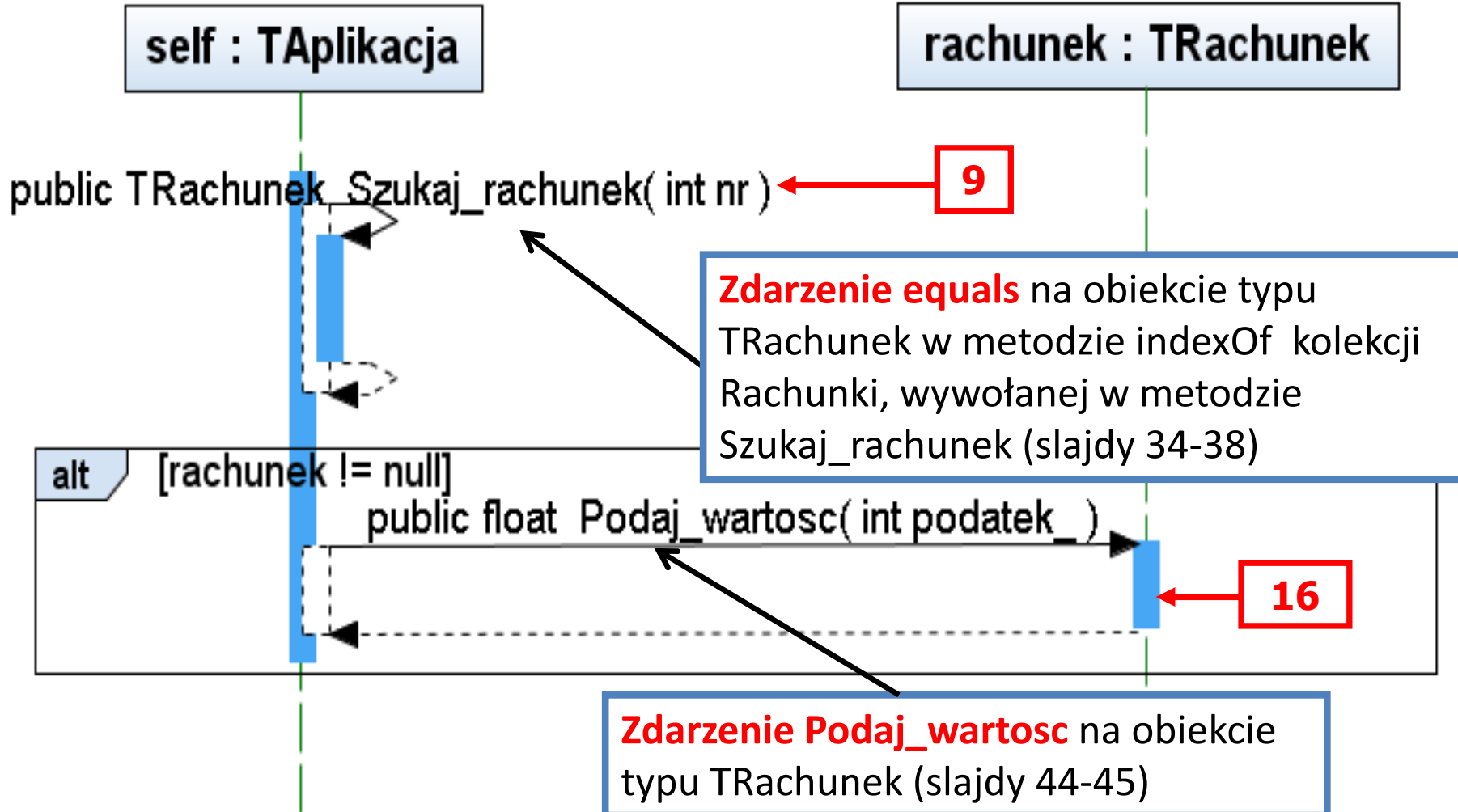
za pomocą diagramu sekwencji i diagramu klas.

Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

Definiowanie kodu metod realizujących przypadek użycia

na podstawie diagramów sekwencji

(15) Obliczanie wartosci rachunku – generowanie zdarzeń **equals** i **Podaj_wartosc** na obiekcie typu TRachunek
(float TApplikacja::Podaj_wartosc(int nr, int podatek_))



//TAplikacja

```
public float Podaj_wartosc (int nr, int podatek_)  
{  
    TRachunek rachunek;  
    rachunek = Szukaj_rachunek(nr);  
    if (rachunek != null)  
        return rachunek.Podaj_wartosc(podatek_);  
    return 0F;  
}
```

(16) float TRachunek::Podaj_wartosc(int podatek_) – metoda zdarzeniowa

self : TRachunek

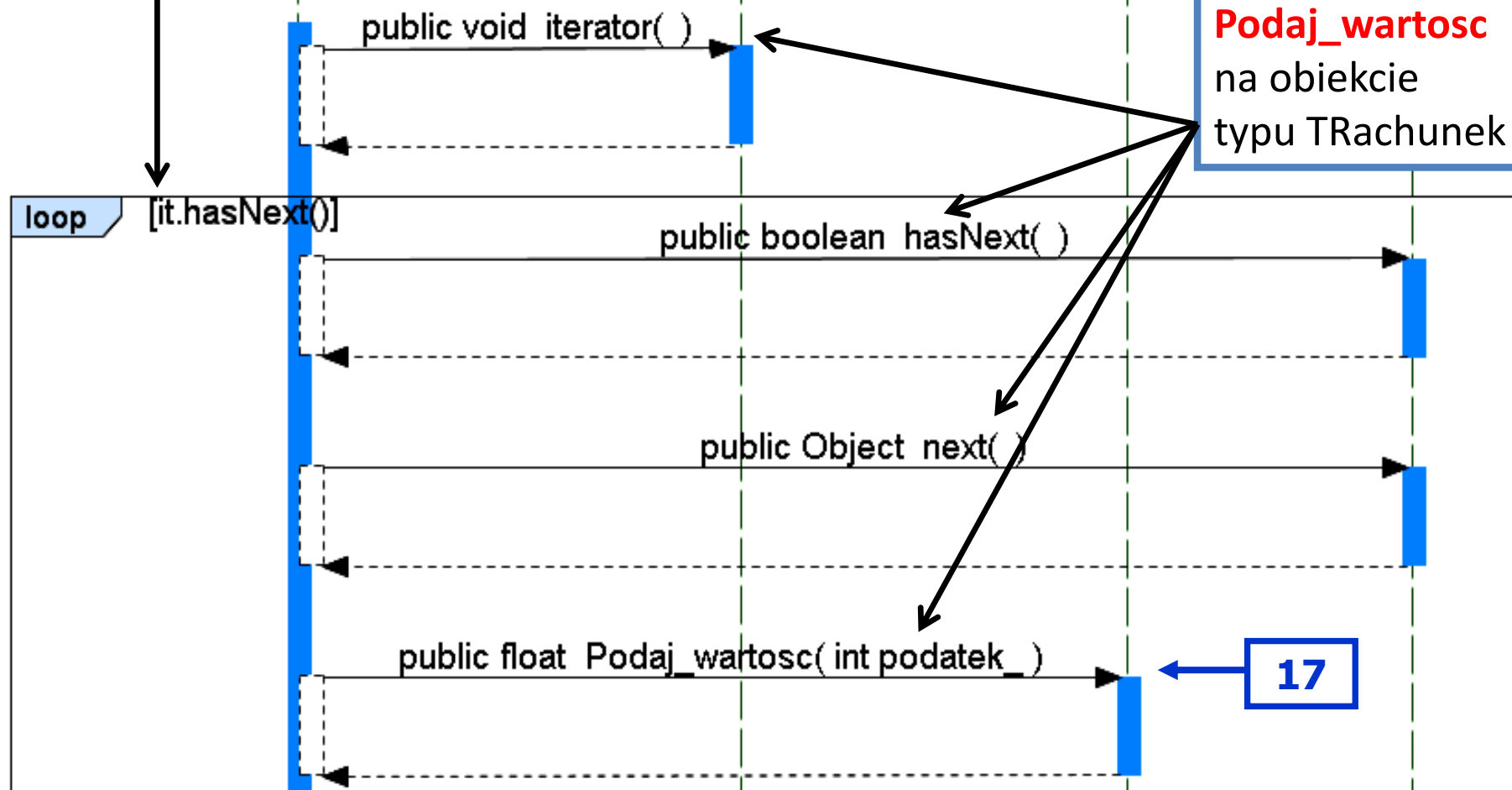
Zakupy : ArrayList

zakup : TZakup

it : Iterator

Dozór zdarzenia **Podaj_wartosc** na obiekcie typu TRachunek

Akcje zdarzenia **Podaj_wartosc** na obiekcie typu TRachunek



```
//metoda zdarzeniowa Podaj_wartosc  
// metody użyte w kodzie metody są akcjami zdarzenia  
//instrukcje warunkowe mogą być użyte jako dozory
```

```
private ArrayList<TZakup> Zakupy = new ArrayList<TZakup>();
```

```
public float Podaj_wartosc (int podatek_  
{  
    float suma=0;  
    TZakup zakup;  
    Iterator <TZakup> it=Zakupy.iterator();  
    while (it.hasNext())  
    { zakup = it.next();  
      suma += zakup.Podaj_wartosc(podatek_  
    }  
    return suma;  
}
```