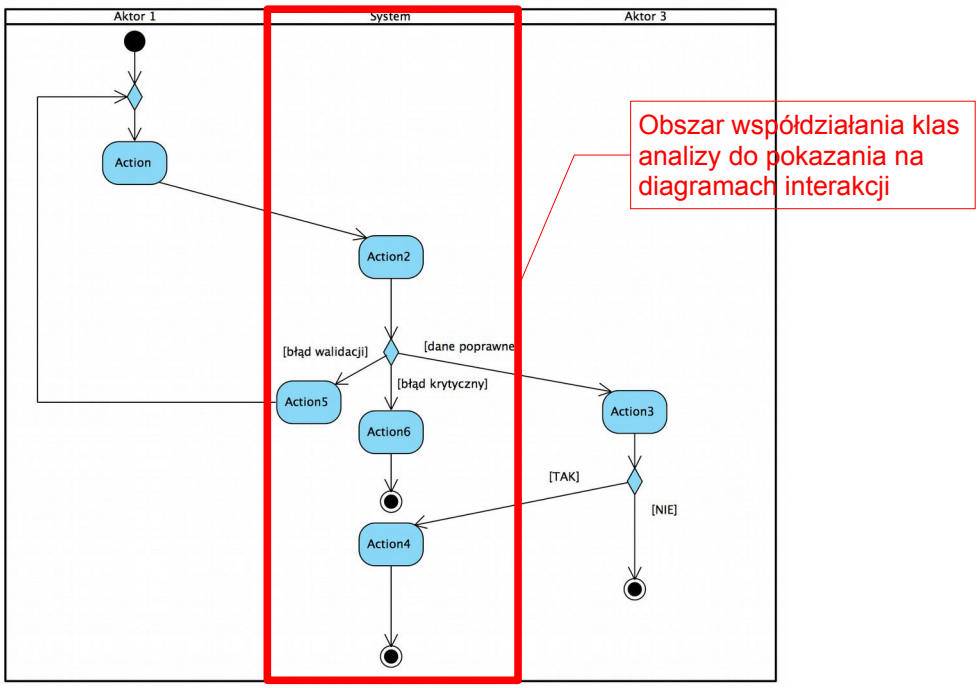
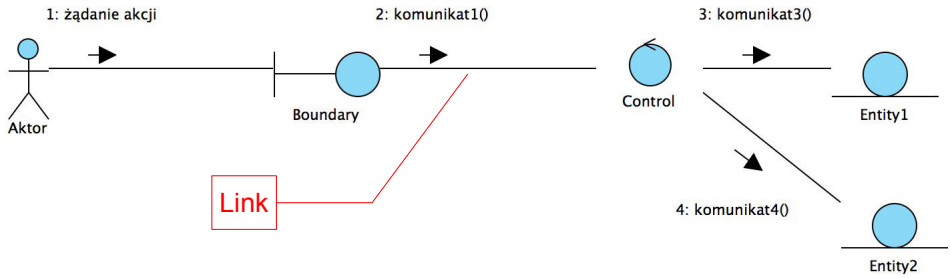
Realizacje przypadków użycia – diagramy interakcji

Diagramy interakcji (komunikacji i sekwencji) służą pokazaniu interakcji między współpracującymi elementami, w naszym przypadku – elementami systemu zidentyfikowanymi wcześniej jako klasy analizy. Na etapie analizy przypadków użycia system traktowaliśmy jako czarną skrzynkę, teraz pokażemy jak działanie systemu ujęte diagramami aktywności będzie realizowane w poszczególnych przypadkach użycia przez klasy analizy z odpowiadających przypadkom użycia diagramów VoPC:



Diagramy komunikacji i sekwencji pokazują ten sam widok na wewnętrzne działanie systemu, jednak są między nimi istotne różnice:

1. Z diagramów komunikacji bezpośrednio wynika, obiekty których klas ze sobą się komunikują. Linki na diagramie aktywności muszą pokrywać się z asocjacjami na diagramach VoPC, czyli diagram komunikacji pozwoli nam zweryfikować związki strukturalne, które wcześniej założyliśmy na VoPC.



2. Diagramy sekwencji nie pokazują wprost linków, jednak można je łatwo wydedukować na podstawie przesyłanych między obiektami komunikatów. Siłą diagramu sekwencji jest uwzględnienie zależności czasowych między działaniami obiektów – oś czasu (jawnie niewidoczna na diagramie) biegnie od góry do dołu diagramu, pokazując, które komunikaty

wysyłane są wcześniej, a które później (co odpowiada numerom komunikatów na diagramie komunikacji).

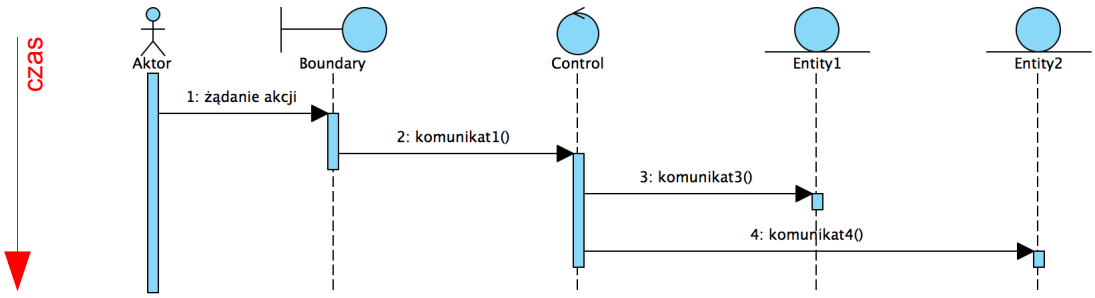


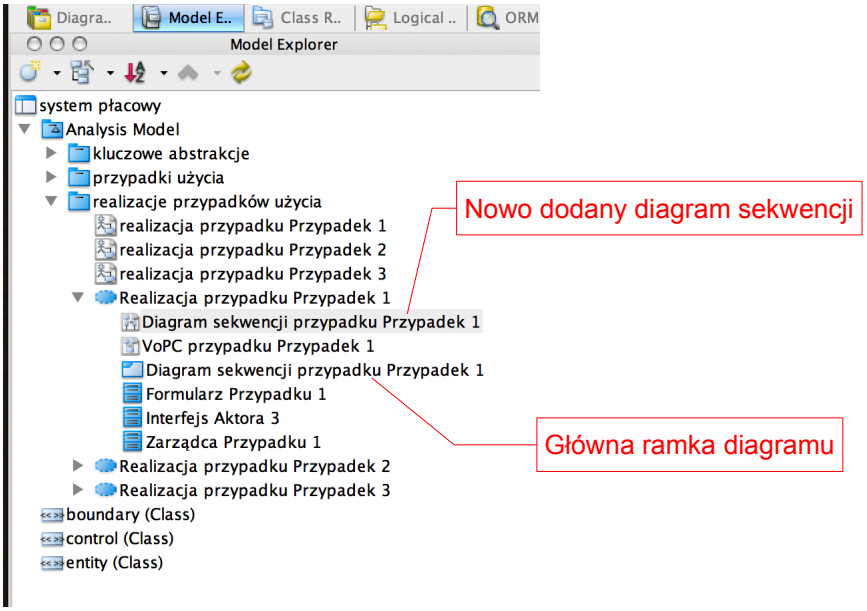
Diagram komunikacji możemy postrzegać jako projekcję diagramu sekwencji wzdłuż osi czasu:

W ramach zajęć do każdej realizacji przypadku użycia dodamy diagram sekwencji, następnie automatycznie wygenerujemy z niego diagram komunikacji. Na sam koniec, na podstawie przygotowanych diagramów, wygenerujemy szkieletowy kod aplikacji, który w rzeczywistych projektach może stanowić początek prac implementacyjnych (pamiętajmy, że jesteśmy na razie na etapie analizy, zidentyfikowane klasy analizy nie są pełnoprawnymi klasami projektowymi).

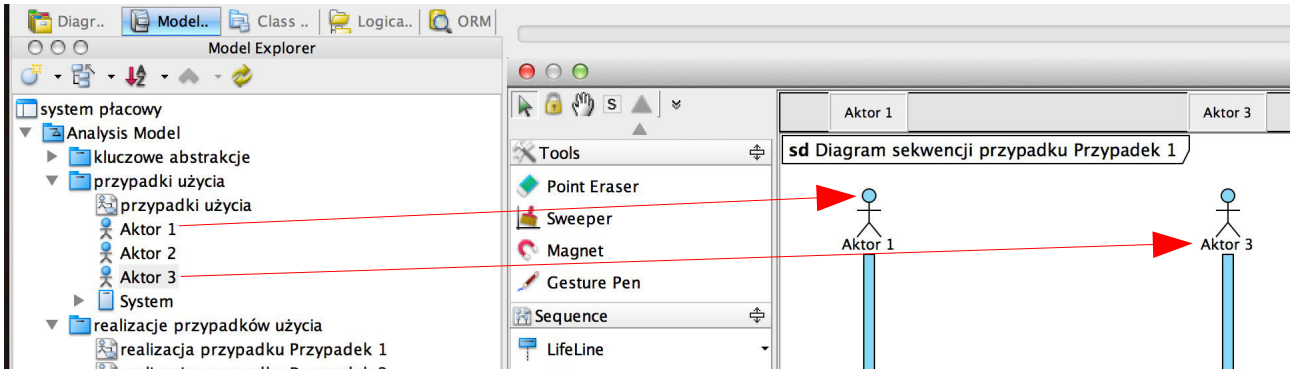
Tradycyjnie, prace rozpoczniemy od wspólnego przygotowania diagramu sekwencji dla jednego z przypadków użycia, następnie, już samodzielnie, przeniesiemy go do modelu Visual Paradigm, postępując zgodnie z instrukcją. Diagramami interakcji dla realizacji drugiego z wybranych przypadków użycia w analogiczny sposób trzeba będzie wykonać samodzielnie.

Model systemu

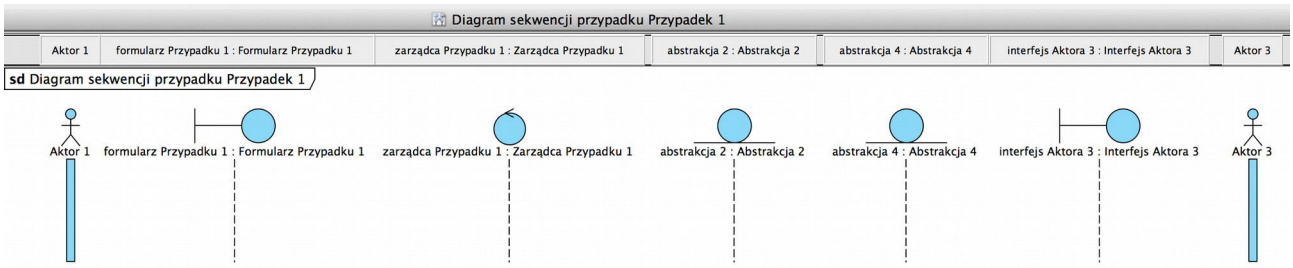
Pierwszym krokiem jest dodanie do realizacji przypadku użycia nowego diagramu sekwencji o nazwie „Diagram sekwencji przypadku <nazwa przypadku>” (obok diagramu VoPC). Zauważmy, że diagramowi towarzyszy jego domyślna ramka, będąca kontenerem pokazywanej sekwencji:



Na diagram przeciągamy z modelu aktorów biorących udział w przypadku użycia, którego realizacją się zajmujemy (nie tworzymy nowych aktorów).

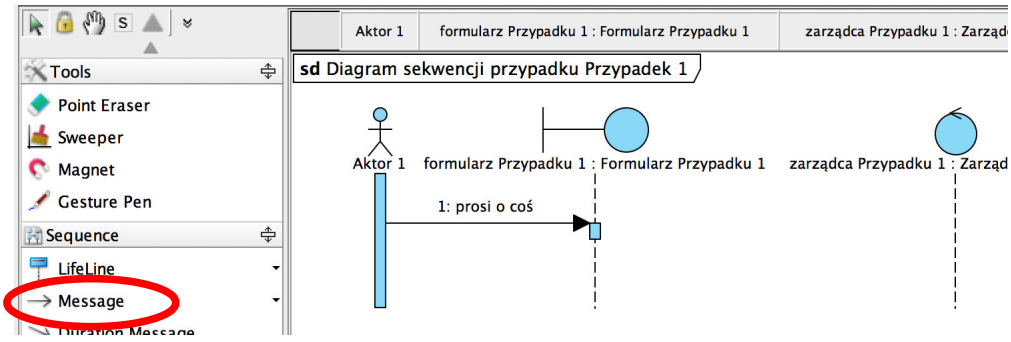


Następnie przeciągamy z realizacji zidentyfikowane wcześniej klasy analizy (jako linię życia - *lifeline*) – najwygodniej umieścić klasy *boundary* tuż przy obsługiwanych przez nie aktorach:



Zauważmy, że aktorzy są przez cały czas aktywni, natomiast obiekty odpowiadające klasom analizy posiadają jedynie „puste" linie życia. Oznacza to, że obiekty wewnątrz systemu aktywują się dopiero, kiedy dostaną sygnał do działania (komunikat – wołanie metody). Czas ich aktywności ogranicza się do czasu wykonania wołanej metody.

Na początek najwygodniej jest zaprojektować działanie wynikające z podstawowego scenariusza przypadku użycia, którego realizacją się zajmujemy. Ponieważ przypadek użycia jest z definicji rozpoczynany przez aktora, pokażmy jak żąda od czegoś od systemu „wyciągając" z niego synchroniczny komunikat (*message*) do obsługującego go obiektu klasy *boundary*:

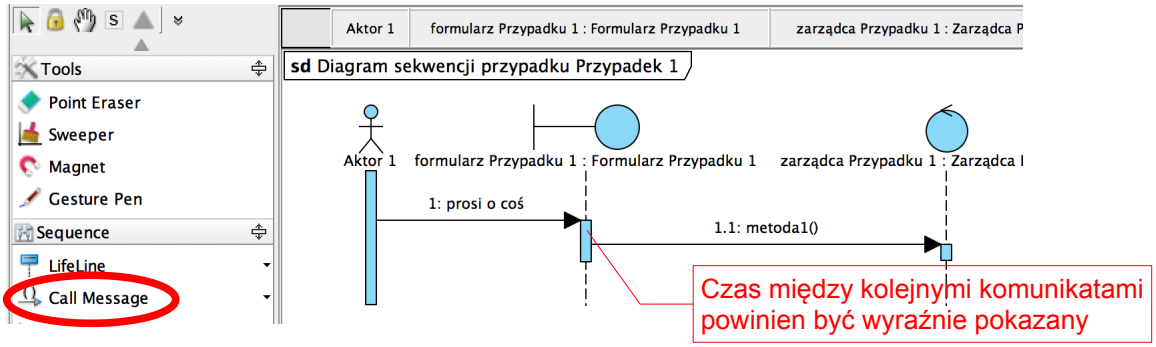


Otrzymanie sygnału z zewnątrz powoduje aktywowanie obiektu klasy *boundary*, który wywołuje odpowiednią metodę na obiekcie klasy *control* (obiekt *boundary* pozostaje aktywny oczekując na powrót sterowania).

Przyjmijmy konwencję nazywania metod stosowania w Javie (*camel case*), np. *zrobCos()*, *zwrocWynik()*, *wykonajBardzoSkomplikowaneObliczenia()*.

Pamiętajmy zasadach komunikacji obiektów poszczególnych klas – dozwolone są jedynie komunikaty między *boundary* a *control*, *control* a *entity* oraz *entity* a *entity*. Wszystkie inne komunikaty będą błędne.

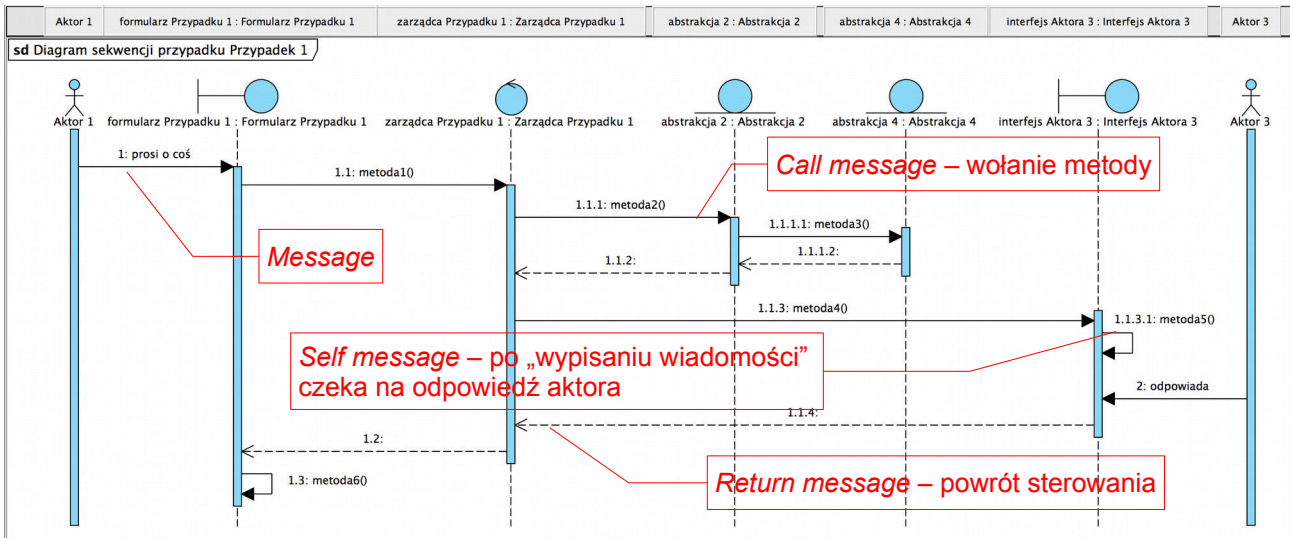
Komunikat między obiektami powinien być typu *call message* (wołanie metody). Metody oznaczamy dodając do nich nawiasy:



Aktor znajduje się poza systemem, więc nie może wołać metody na obiektach wewnątrz systemu. Może jedynie wykonać jakieś działanie, które zostanie przez system rozpoznane (zdarzenie wykryte przez obiekt *boundary*), a następnie odpowiednio zinterpretowane i obsłużone.

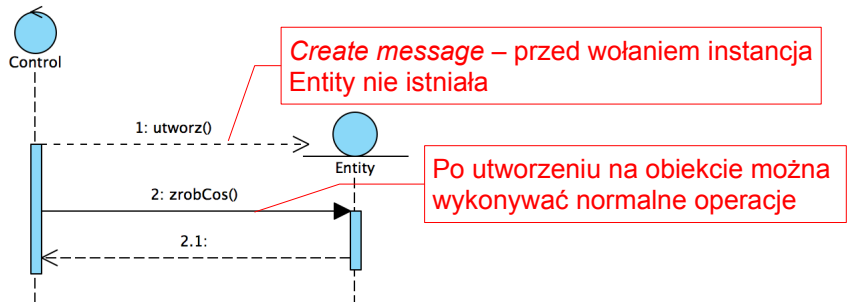
Dalsze zachowanie wewnątrz systemu jest zależne od przypadku użycia, którego realizacją się zajmujemy (nadal obsługujemy wyłącznie scenariusz podstawowy). Po wykonaniu metody z wołania synchronicznego dodajemy zawsze powrót sterowania (*return message*), co oznacza zwrócenie sterowania do obiektu wołającego. Nie ma większego sensu nadawanie nazw powrotom sterowania:

Dbajmy o odpowiednie „rozsunięcie" komunikatów (włącznie z powrotami sterowania) wzdłuż osi czasu, aby nie nachodziły na siebie w tym samym momencie, ani kolejność ich wykonanie nie była zamienione miejscami w czasie.

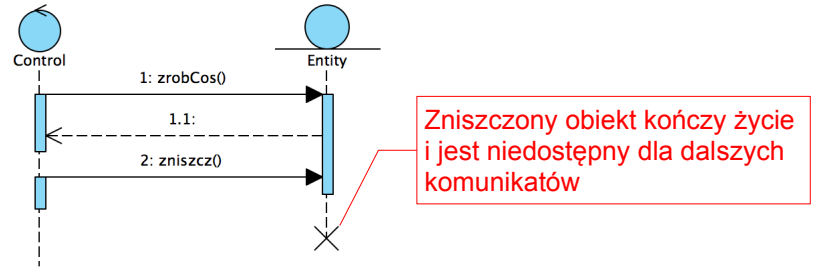


Komunikacja z aktorem na obiektach *boundary* pokazana jest przez wołanie metody na samym sobie (*self message*). Wynika to z tego, że aktor jest względem systemu całkowicie niezależny i asynchroniczny, więc system nie może mu zwrócić sterowania. Może jedynie „wypisać" coś na obiekcie boundary i czekać na reakcję aktora (która nastąpi lub nie).

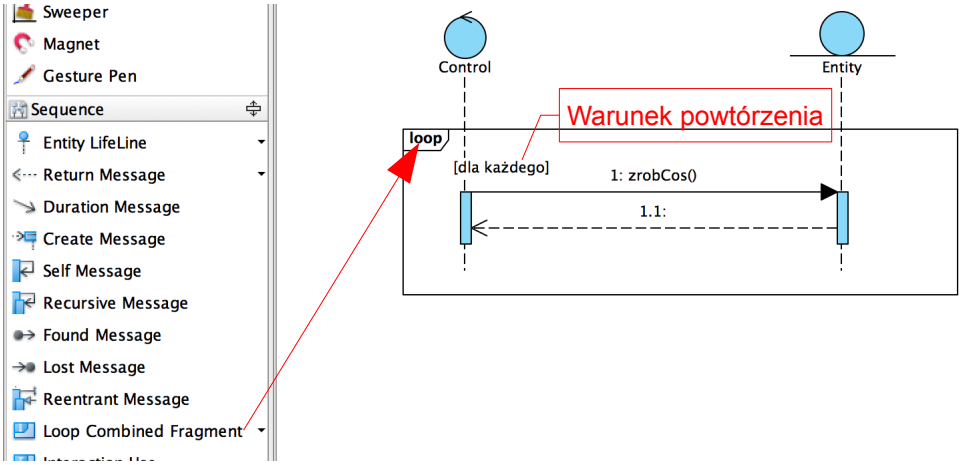
Jeżeli potrzebujemy utworzyć nowy obiekt (zamiast wysyłać komunikat do istniejącego), możemy posłużyć się specjalnym komunikatem typu *create* (oznaczającym na przykład wołanie konstruktora klasy tego obiektu):



Analogicznie, komunikat *destroy* pozwala na usunięcie (zniszczenie) obiektu:

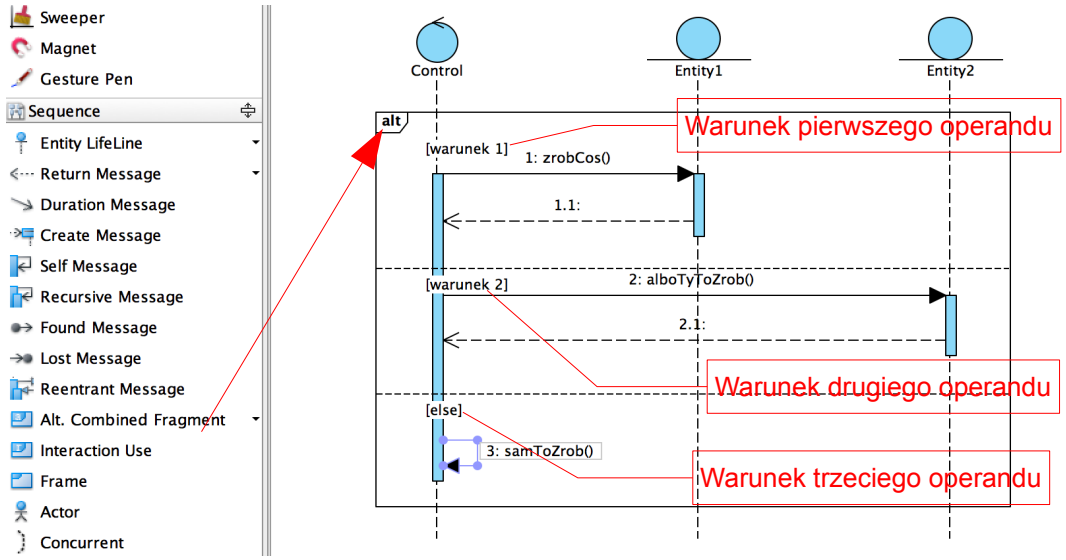


Jeżeli w ciągu zdarzeń pojawia się wykonanie powtarzalne (pętla), możemy to pokazać przez fragment *loop* (*Loop Combined Fragment*):



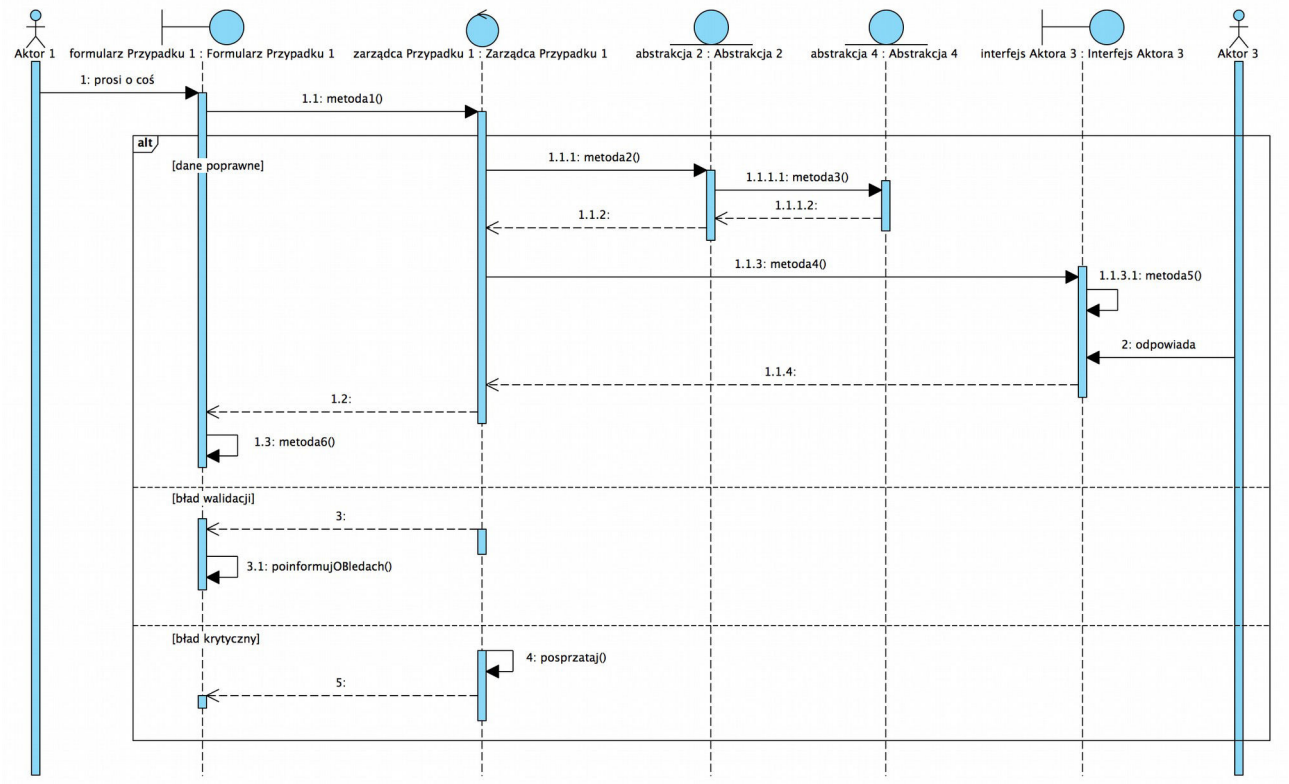
Warunki poszczególnych operandów ustawiamy przez menu kontekstowe fragmentu > *Operand* > *Edit Operand* i podając *Constrain*t (podobnie dodajemy i usuwamy operandy).

Natomiast wykonanie warunkowe (wynikające na przykład z ciągów alternatywnych) pokazujemy przez fragment *alt* (*Alt. Combined Fragment*). Poniższy rysunek pokazuje wykonanie w rodzaju *if-elseif-else*:

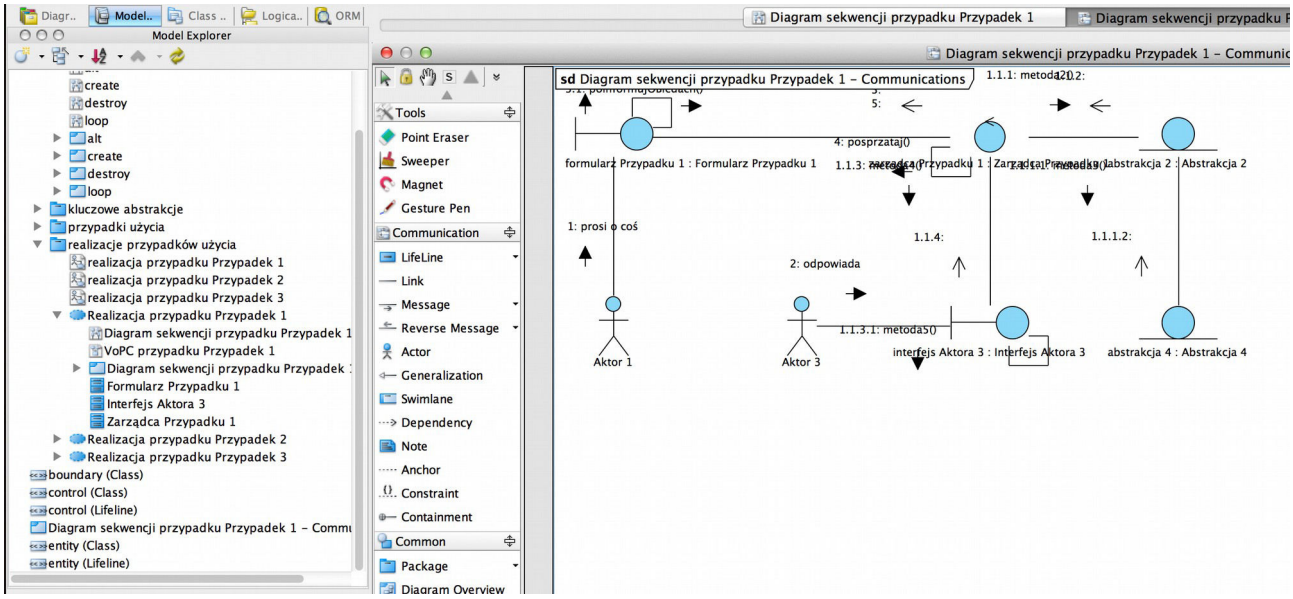


Istnieje więcej typów specjalnych fragmentów, jednak raczej nie będą przydatne przy modelowaniu naszego przykładowego systemu.

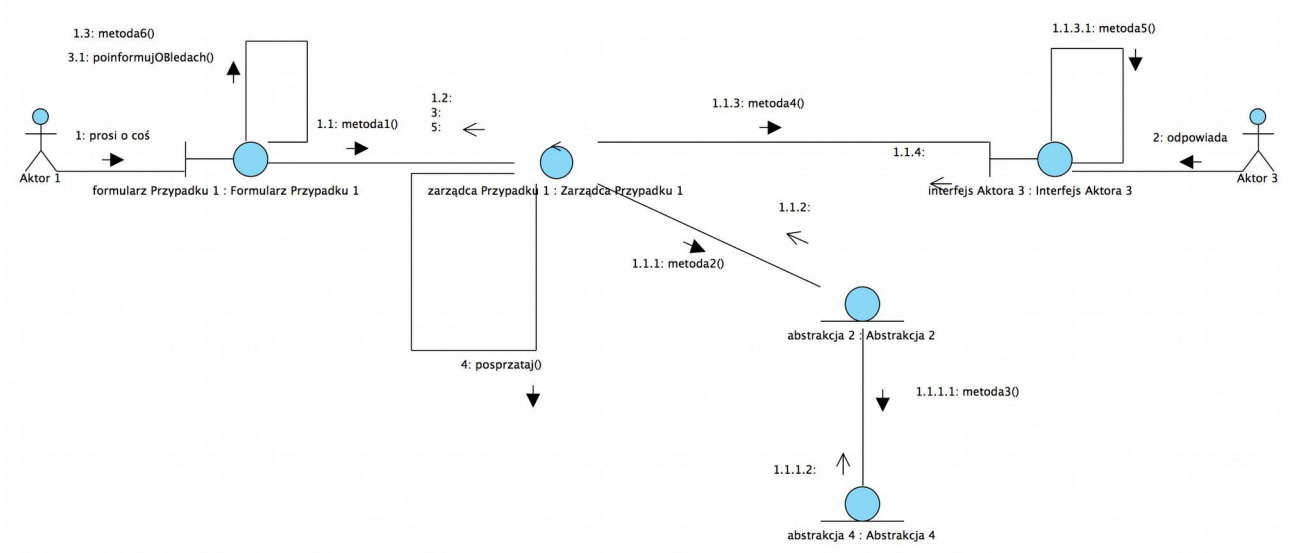
Kiedy umiemy już pokazywać na diagramie sekwencji wykonanie warunkowe, możemy obsłużyć wszystkie sytuacje inne niż wykonanie standardowe (rozgałęzienia i ciągi alternatywne):



Zakładając, że diagram sekwencji jest kompletny, możemy z niego wygenerować automatycznie diagram komunikacji. To bardzo prosta operacja – z menu kontekstowego diagramu sekwencji (w edytorze, nie modelu) wybieramy *Sychronize to Communication Diagram*. I gotowe:



Pozostaje nadać mu nazwę zgodną z naszą konwencją („Diagram komunikacji przypadku <nazwa przypadku>”) i oczywiście – poprawić jego czytelność:

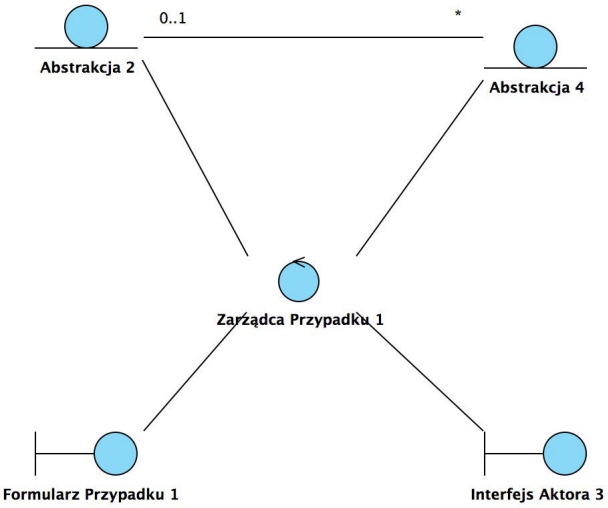


> Wygenerowany w ten sposób diagram komunikacji pozostaje w relacji do oryginalnego diagramu sekwencji – jest jedynie jego projekcją i nie stanowi samodzielnego bytu w modelu.

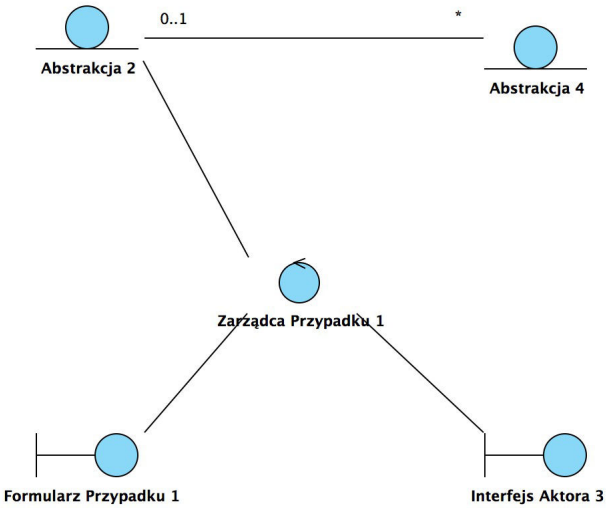
Jak wielokrotnie wspominałem, diagram komunikacji kładzie nacisk na połączenia między obiektami (linki na diagramie). Każdy link między obiektami klas *entity* a *control* odpowiada asocjacji na diagramie VoPC.

> Asocjacje między klasami *entity* są założeniami z poziomu modelu dziedzinowego i nie muszą być spełnione na diagramach interakcji.

Sprawdźmy więc VoPC dla realizacji tego przypadku użycia:



Wygląda na to, że nasza realizacja nie zakłada komunikacji między klasą *control* (*Zarządca Przypadku 1*) a klasą entity *Abstrakcja 4*. Czyli ta asocjacja jest zbędna, usuwamy ją z VoPC:



Weryfikujemy diagramy interakcji (najłatwiej – komunikacji) z VoPC w odpowiednich realizacjach i usuwamy zbędne, a dodajemy brakujace asocjacje.

Ostatnim krokiem (kiedy już wszystkie diagramy sekwencji są gotowe) jest sprawdzenie generacji szkieletu kodu systemu w oparciu o model. Z paska narzędzi rozwijamy *Code > Instant Generator*, a następnie język, w którym pracujemy. Pojawia się okno dialogowe, w którym konfigurujemy parametry generacji kodu. W zakładce *Model Elements* zaznaczamy, które elementy modelu mają zostać wykorzystane do generacji kodu (u nas – wszystkie).

Podajemy ścieżkę katalogu, do którego mają być zapisane pliki i naciskamy *Generate* (podgląd dostępny jest przez *Preview*). Ostatnim etapem ćwiczenia jest zapoznanie się z kodem wygenerowanym automatycznie przez Visual Paradigm.