Krzysztof Regulski
AGH, WIMiIP

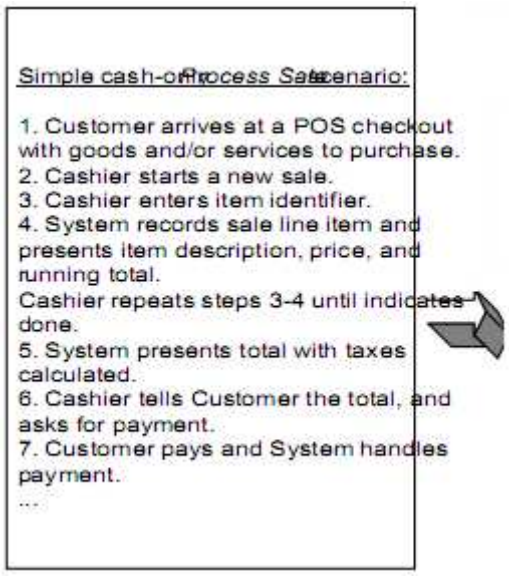
DIAGRAM SEKWENCJI

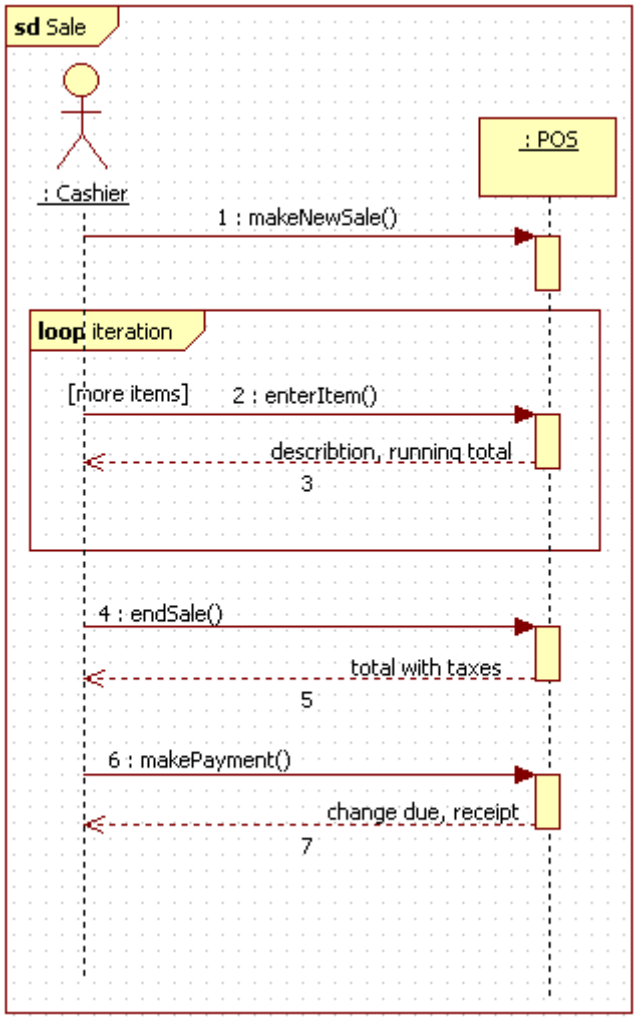
1) Zastosowanie

- Diagram sekwencji ilustruje zdarzenia wejściowe i wyjściowe, wskazuje jakie zdarzenia wywołują zewnętrzni aktorzy. Diagram przedstawia interakcje aktorów z systemem i operacje wywołane przez nich. Samo działanie systemu jest mniej ważne, system traktujemy jak czarną skrzynkę, istotne są zdarzenia, które przekraczają granice system-aktor.
- Na diagramie sekwencji ważna jest kolejność zdarzeń. Czas płynie „w dół".
- Diagramy sekwencji nie zajmują się operacjami na danych, czy dostępem do danych. Ważny jest przepływ sterowania.

2) Konstrukcja

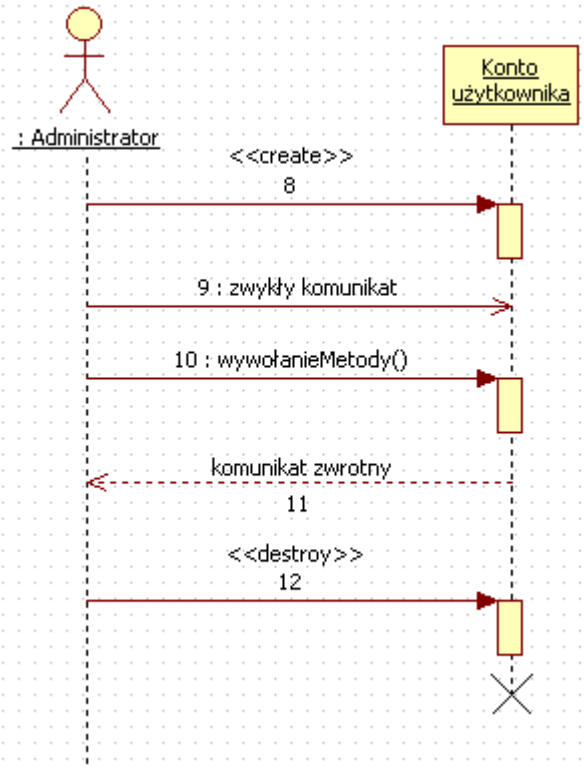
- Diagramy sekwencji budujemy dla poszczególnych przypadków użycia.
- Tam, gdzie to możliwe, należy używać klas i operacji zdefiniowanych w diagramie klas.

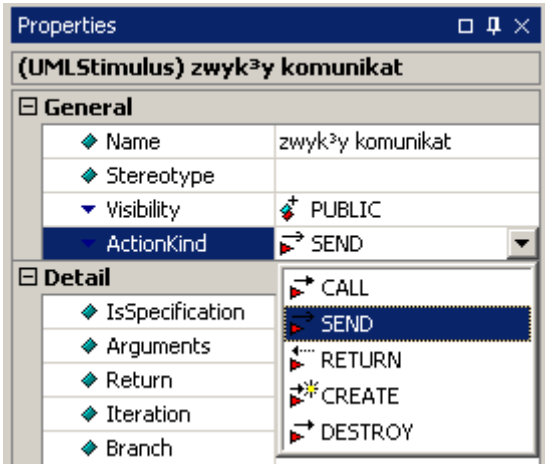




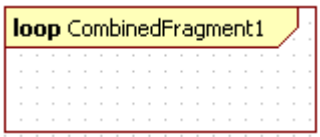
► Komunikaty mogą być różnego typu:

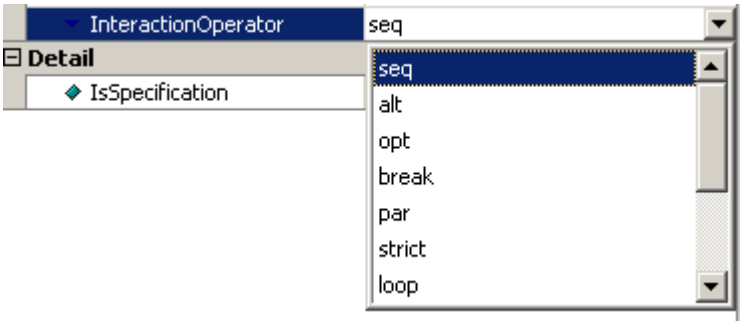
- SEND – komunikat standardowy, asynchroniczny (po wysłaniu nie czeka na odpowiedź, ale kontynuuje działanie)
- CALL – komunikat wywołujący metodę innej klasy, najczęściej synchroniczny
- RETURN – komunikat zwrotny
- CREATE – komunikat tworzący obiekt
- DESTROY – komunikat usuwający obiekt z systemu





► Można wydzielać fragmenty z diagramu sekwencji stosując różne operatory:
- seq, alt, opt, break, par, stricte, loop, region, neg, assert, ignore, consider





[ZAŁĄCZNIK]

► Zastosowanie poszczególnych operatorów :

(Russ, Hamilton, *Learning UML 2.0*, O'Reilly, 2006)

Type	Parameters	Why is it useful?
ref	None	Represents an interaction that is defined elsewhere in the model. Helps you manage a large diagram by splitting, and potentially reusing, a collection of interactions. Similar to the reuse modeled when the `<<include>>` use case relationship is applied.
assert	None	Specifies that the interactions contained within the fragment box must occur exactly as they are indicated; otherwise the fragment is declared invalid and an exception should be raised. Works in a similar fashion to the `assert` statement in Java. Useful when specifying that every step in an interaction must occur successfully, i.e., when modeling a transaction.
loop	min times, max times, [guard_condition]	Loops through the interactions contained within the fragment a specified number of times until the guard condition is evaluated to false. Very similar to the Java and C# `for(..)` loop. Useful when you are trying execute a set of interactions a specific number of times.
break	None	If the interactions contained within the break fragment occur, then any enclosing interaction, most commonly a loop fragment , should be exited. Similar to the `break` statement in Java and C#.
alt	[guard condition1] ... [guard_condition2] ... [else]	Depending on which guard condition evaluates to true first, the corresponding sub-collection of interactions will be executed. Helps you specify that a set of interactions will be executed only under certain conditions. Similar to an `if(..) else` statement in code.
opt	[guard_condition]	The interactions contained within this fragment will execute only if the guard condition evaluates to true. Similar to a simple `if(..)` statement in code with no corresponding `else`. Especially useful when showing steps that have been reused from another use case's sequence diagrams, where `<<extend>>` is the use case relationship.
neg	None	Declares that the interactions inside this fragment are not to be executed, ever. Helpful if you are just trying to mark a collection of interactions as not executed until you're sure that those interactions can be removed. Most useful if you happen to be lucky enough to be using an Executable UML tool where your sequence diagrams are actually being run. Also can be helpful to show that something cannot be done, e.g., when you want to show that a participant cannot call `read( )` on a socket after `close( )`.Works in a similar fashion to commenting out some method calls in code.
par	None	Specifies that interactions within this fragment can happily execute in parallel. This is similar to saying that there is no need for any thread-safe locking required within a set of interactions.
region	None	Interactions within this type of fragment are said to be part of a critical region. A critical region is typically an area where a shared participant is updated. Combined with parallel interactions, specified using the par fragment type, you can model where interactions are not required to be thread- or process-safe (par fragment) and where locks are required to prevent parallel interactions interleaving (region fragment). Has similarities synchronized blocks and object locks in Java.