Sequence Diagrams

Version 060201
Sequence Diagrams

- Sequence Diagrams are
  - simple
  - powerful
  - readable
  - used to describe interaction sequences

- History
  - Has been used for a number of years informally
  - Standardized in 1992 in Z.120 (Message Sequence Charts - MSC)
  - Last major revision of MSC is from 1999 (called MSC-2000)
  - Formal semantics of MSC-96 is given in Z.120 Annex B
  - Included in UML from 1999, but in a rather simple variant
Purpose

- Emphasizes the interaction between objects indicating that the interplay is the most important aspect
  - Often only a small portion of the total variety of behavior is described to improve the individual understanding of an interaction problem

- Sequence Diagrams are used to ...
  - document protocol situations,
  - illustrate behavior situations,
  - verify interaction properties relative to a specification,
  - describe test cases,
  - document simulation traces.
The example context: Dolly Goes To Town

- Dolly is going to town and
  - wants to subscribe for bus schedules back home
  - given her current position
  - and the time of day.
  - The service should not come in effect until a given time in the evening
The informal architecture
Lifeline (MSC: Instance) – the “doers”

Lifeline head with name the lifeline itself is anonymous, but its type is ServiceBase

Lifeline ordered from top to bottom
(Simple) Sequence Diagram

- Messages have one send event, and one receive event.
  - The send event must occur before the receive event.
  - The send event is the result of an Action
- Events are strictly ordered along a lifeline from top to bottom
How many global traces are there in this diagram?

- The only invariants:
  - Messages have one send event, and one receive event. The send event must occur before the receive event.
  - Events are strictly ordered along lifeline

How many?
- 1, 2, 3, 4, 5, 6,..?

-dependent!
Really counting the traces ...
Asynchronous messages: Message Overtaking

- asynchronous communication = when the sender does not wait for the reply of the message sent
- Reception is normally interpreted as consumption of the message.
- When messages are asynchronous, it is important to be able to describe message overtaking.

Diagram:
- Notice message to/from environment
- Sending `OnWeb` before sending `Info`
- Receiving `OnWeb` after receiving `Info`
The context of a Sequence Diagram

- The context is a Classifier with Composite Structure (of properties)
  - Properties (parts) are represented by Lifelines
  - Generic Parts of Collaborations must be bound to concrete Parts
  - Concrete Parts of Classes can be Lifelines directly
- In MSC (Message Sequence Charts) context is an “MSC document”
- The concept of a context with internal structure leads to an aggregate hierarchy of entities (parts)
  - We exploit this through the concept of Decomposition
In Rational Software Modeler (Class+comp. struct.)

- model explorer
- model
- classes
- parts / properties
- operations (behaviors)
- diagrams
- composite structure
Sequence Diagrams in SeDi
Synchronizing interaction

- **Authorization**: `sd`
- **ServiceUser**: `:ServiceUser`
- **ServiceBase**: `:ServiceBase`
- **ServiceTerminal**: `:ServiceTerminal`

**Message Flow**:
- **Code**: `Code`
- **OnWeb**: `OnWeb`
- **OK**: `OK`

**Annotations**:
- Method call message
- Execution specification
- Reply
Basic Sequence Diagrams Summary

- We consider mostly messages that are **asynchronous**, the sending of one message must come before the corresponding reception
- UML has traditionally described **synchronizing** method calls rather than asynchronous communication
- The events on a lifeline are **strictly ordered**
- The **distance** between events is not significant.
- The context of Interactions are **classifiers**
More structure (UML 2.0 from MSC-96)

- A lifeline (within an interaction) may be detailed in a decomposition
- Dynamic **creation and destruction** of lifelines
- interaction **uses** – such that Interactions may be referenced within other Interactions
- combined fragments – combining Interaction fragments to express alternatives, parallel merge and loops
- better overview of combinations – High level Interactions where Lifelines and individual Messages are hidden
- gates – flexible connection points between references/expressions and their surroundings
References

interaction use

sd GoHome

:ServiceUser

:ServiceBase

:ServiceTerminal

ref

GoHomeSetup

loop

ref

GoHomeInvocation

ref

GoHomeDismantle
Combined fragments of Interaction

- MSC-96: “inline expressions”
- UML 2.0: “combined fragments”
- We want to express
  - choices: alternative, option, break
  - parallel merge
  - loops
- We also want to add other operators
  - negation
  - critical region
  - assertion
- Other suggested operators that will not come in UML 2.0
  - interrupt
  - disrupt
Combined fragment example

```
sd GoHomeInvocation

:ServiceUser   :Clock   :ServiceBase   :ServiceTerminal

InvocationTime

ref
FindLocation
TransportSchedule

loop

ref
ScheduleIntervalElapsed
FindLocation
TransportSchedule

alt

ref
GetTransportSchedule
TransportSchedule

ref
TransportSchedule

FetchSchedule
```
Interaction Overview

- similar to activity diagram
- interaction use
- start
- decision
- flow line
- inline diagram

End
Gates

```
\textbf{sd} GoHomeSetup
\begin{itemize}
  \item \textbf{:ServiceUser}
  \item \textbf{:ServiceBase}
  \item \textbf{:ServiceTerminal}
\end{itemize}

\textbf{opt}
\begin{itemize}
  \item \textbf{ref} Authorization
    \textbf{ref} FindLocation
    \textbf{ref} SetHome
    \textbf{ref} SetInvocationTime
    \textbf{ref} SetTransportPreferences
\end{itemize}

\textbf{sd} Authorization
\begin{itemize}
  \item \textbf{:ServiceBase}
  \item \textbf{:ServiceTerminal}
\end{itemize}

\textbf{Code}
\textbf{OK}
\textbf{OnWeb}
\textbf{info}
\textbf{OK}
```