

Refinement II

Inherent nondeterminism versus underspecification

And weak sequencing

Ketil Stølen

Outline

- Two kinds of nondeterminism
 - Underspecification
 - Inherent (explicit) nondeterminism
- The need for both alt and xalt
- Semantics in the general case
- Refinement in the general case

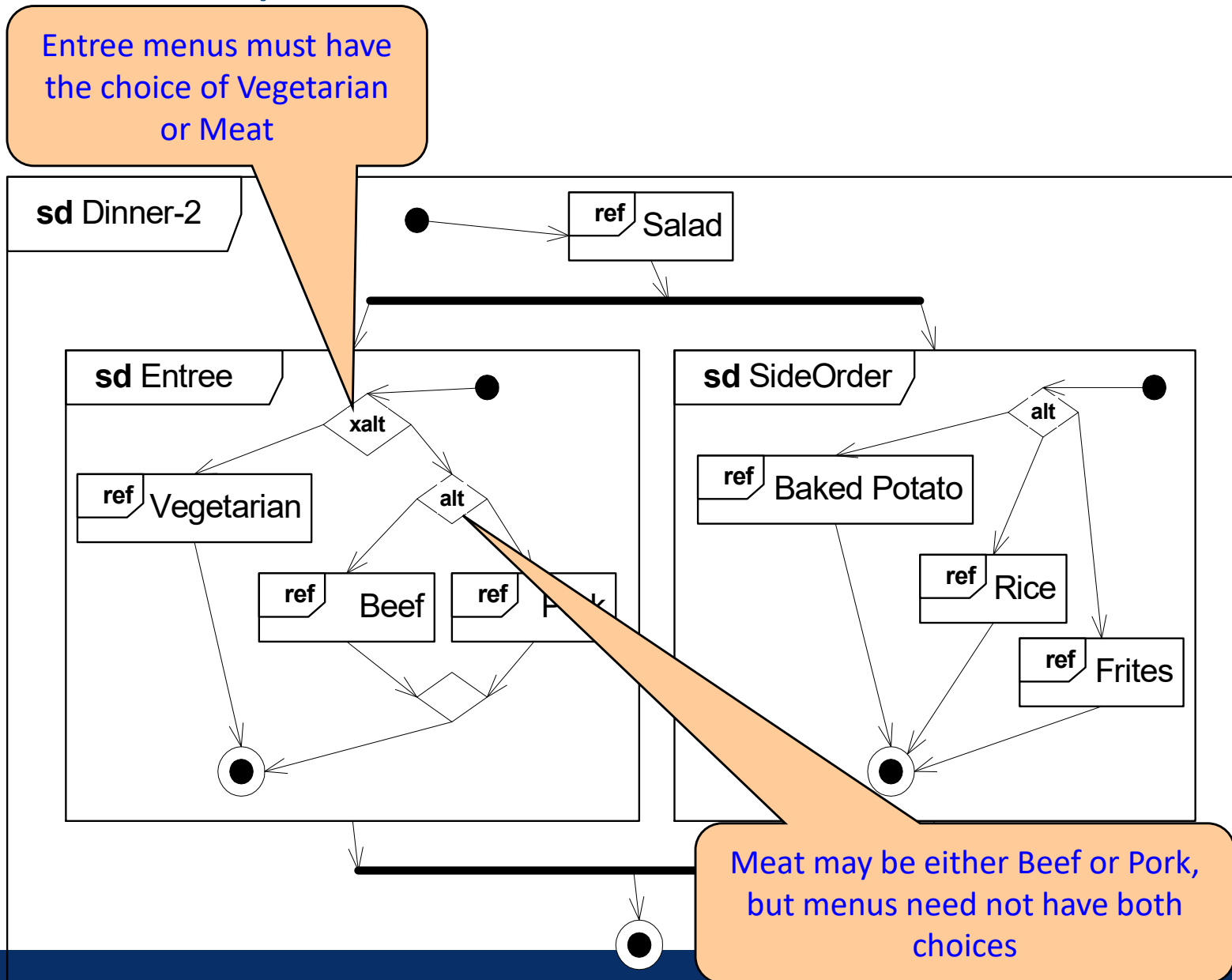
Underspecification and inherent nondeterminism

- Underspecification:
 - Several alternative behaviours are considered equivalent (serve the same purpose)
- Inherent nondeterminism:
 - Alternative behaviours that must all be possible for the implementation
- These two should be described differently!

The need for both **alt** and **xalt**

- Potential non-determinism captured by **alt** allows abstraction and inessential non-determinism
- Inherent or explicit non-determinism captured by **xalt** characterizes non-determinism that must be reflected in every correct implementation in one way or another

Restaurant example with both **alt** and **xalt**

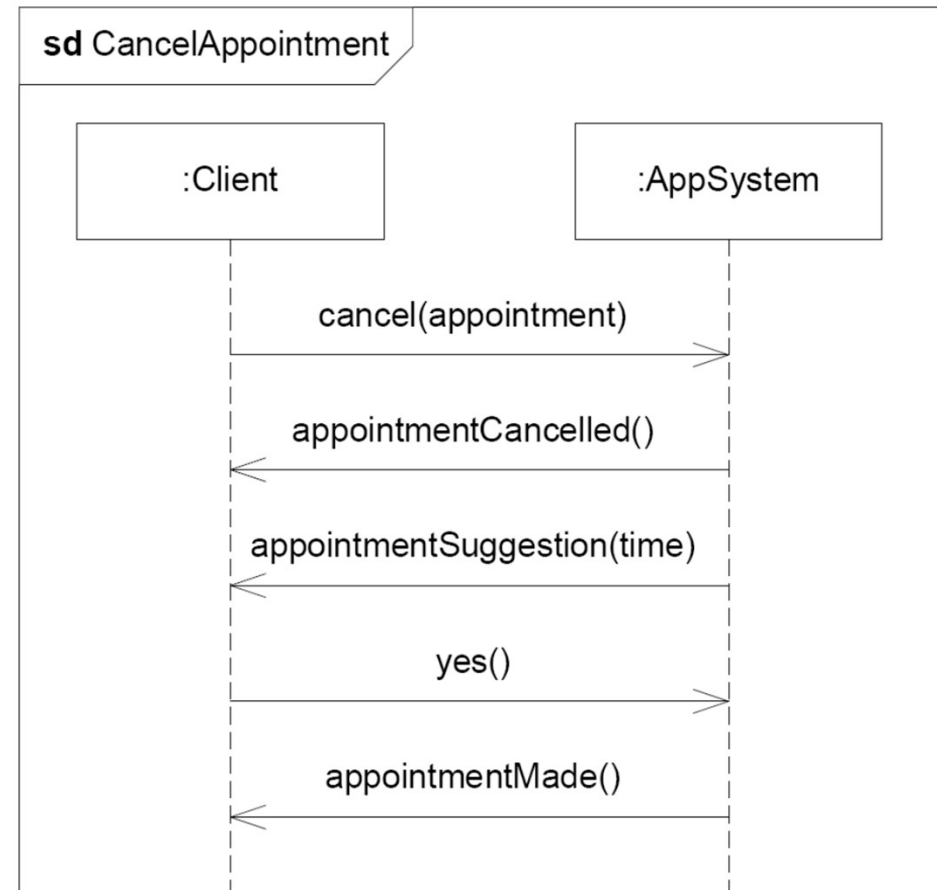


Example: an appointment system

- A system for booking appointments used by e.g. dentists
- Functionality:
 - MakeAppointment: The client may ask for an appointment
 - CancelAppointment: The client may cancel an appointment

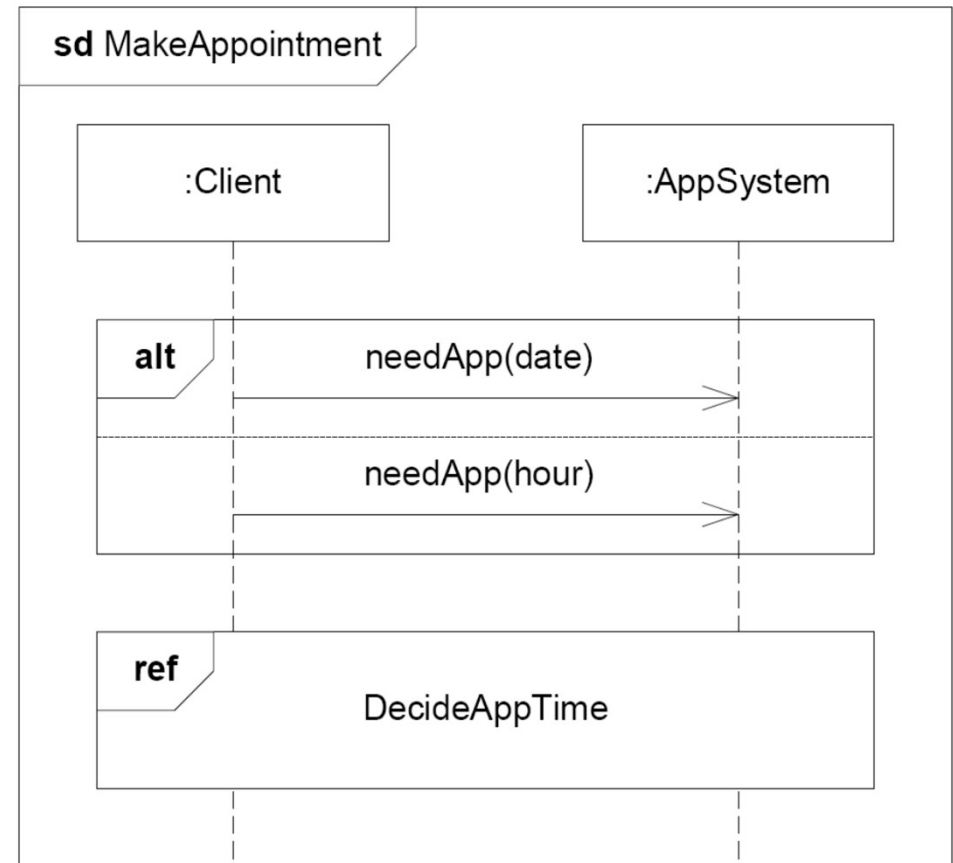
CancelAppointment

- This specification has two positive traces
- Whether reception of `appointmentCancelled()` occurs before or after sending of `appointmentSuggestion(...)` is not important
- Underspecification due to weak sequencing



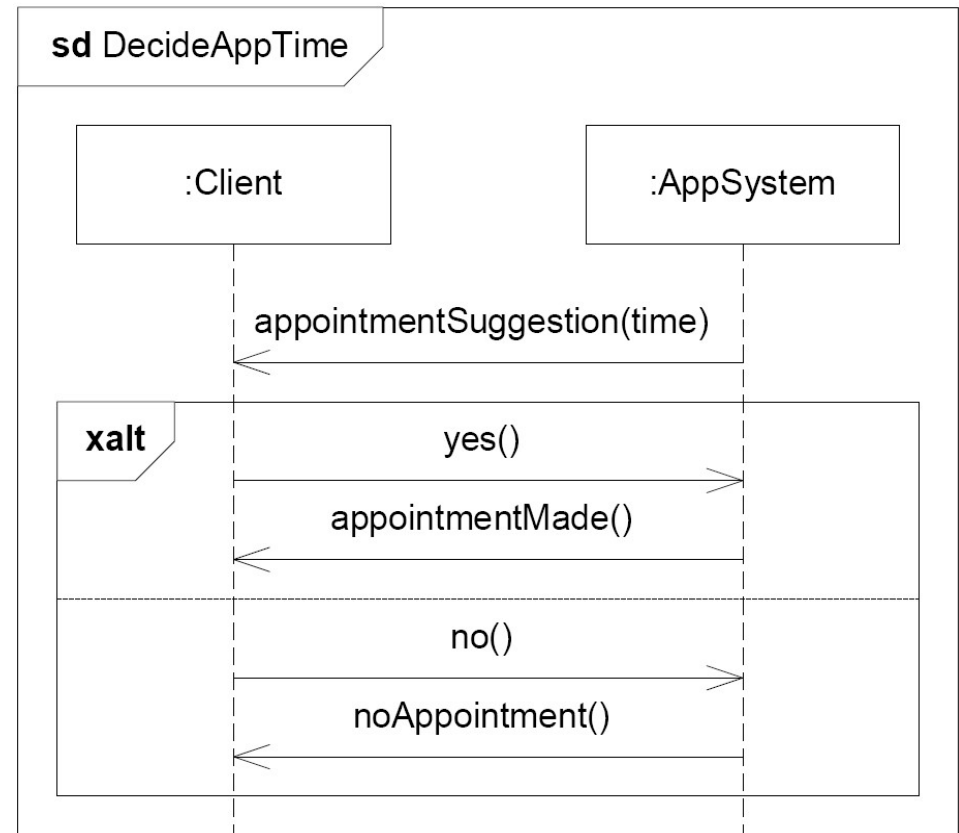
MakeAppointment

- May ask for either a specific date or a specific hour of the day (e.g. in the lunch break)
- The system is not required to offer both alternatives
- Underspecification expressed by the **alt** operator



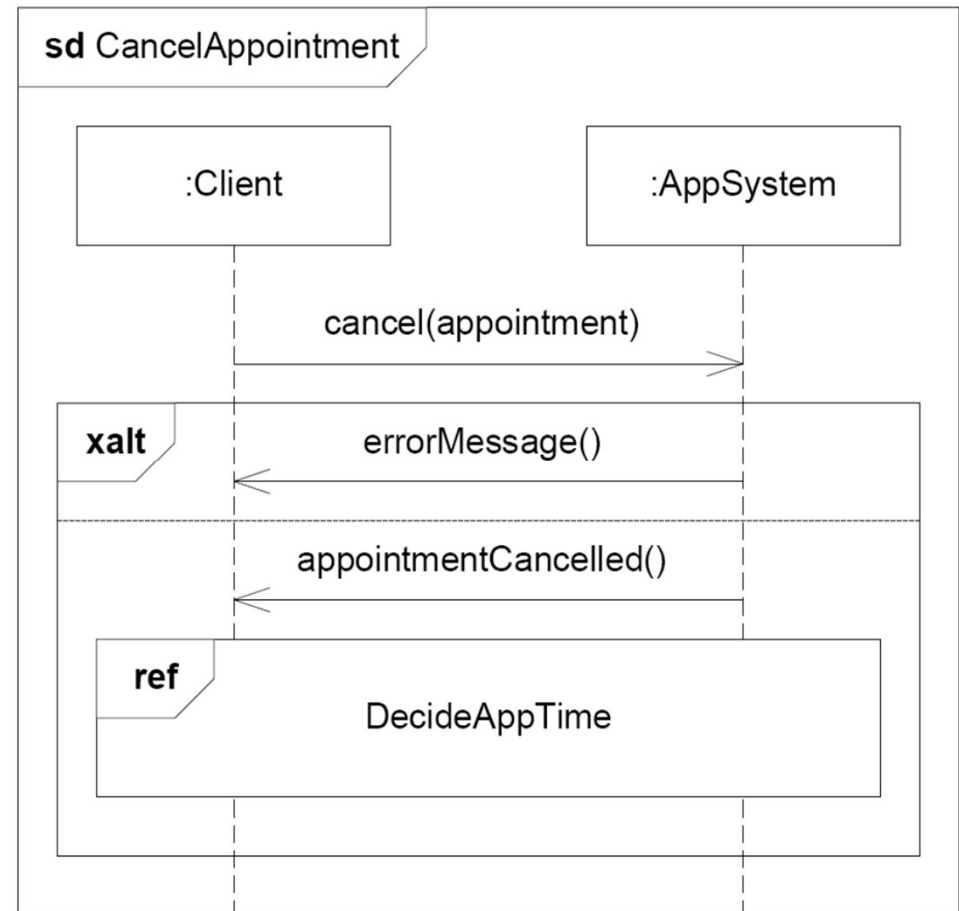
DecideAppTime

- The system must be able to handle *both* yes() and no() as reply messages from the client
- This is *not* underspecification
- Therefore the alternatives are expressed by the **xalt** operator



CancelAppointment - revised

- The condition for choosing `errorMessage()` or `appointmentCancelled()` is not shown
- Both alternatives should be possible
- The choice is made by the system



Use of **alt** versus **xalt**

The crucial question when specifying alternatives:

- Do these alternatives represent similar traces in the sense that implementing only one is sufficient?

When to use **alt**

- Use **alt** to specify alternatives that represent similar traces i.e. to model
 - underspecification

When to use **xalt**

- Use **xalt** to specify alternatives that must all be present in an implementation, i.e. to model
 - inherent nondeterminism, as in the specification of a coin toss or a password generator
 - alternative traces due to different inputs that the system must be able to handle (as in DecideAppTime)
 - alternative traces where the conditions for these being positive are abstracted away (as in revised version of CancelAppointment)

Semantics

So far I have told you that the semantics of a sequence diagram is an interaction obligation

- In the general case, this is not sufficiently expressive

Semantics – general case

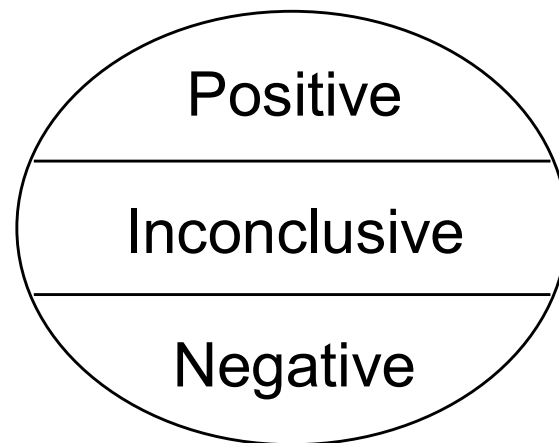
- The semantics of a sequence diagram without occurrences of **xalt** is a set of a single interaction obligation

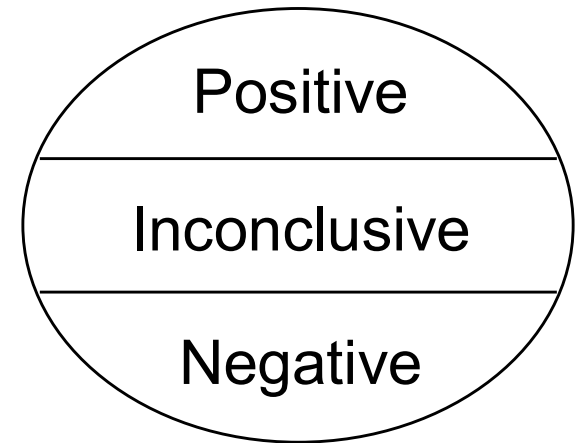
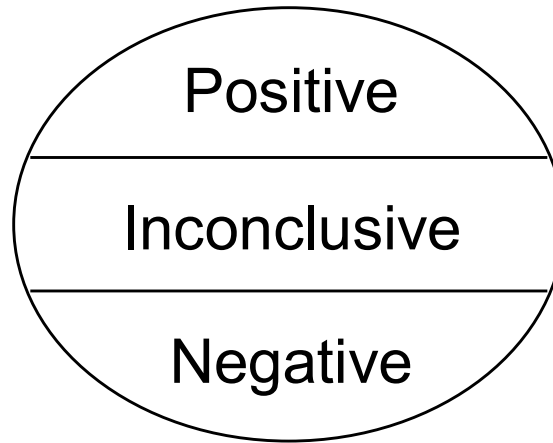
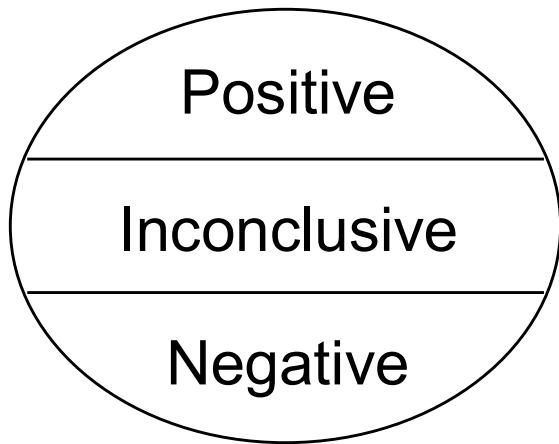
$\{ (p,n) \}$

- The semantics of a sequence diagram with occurrences of **xalt** is a set of arbitrarily many interaction obligations

$\{ (p1,n1), (p2,n2), \dots, (pK,nK) \}$

alt





xalt

