Development of Component- and Service-Oriented Software Architectures with UML and UML-RT — Sequence Diagrams and statecharts —

Ingolf H. Krueger
ikrueger@ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA
Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts
Components and their Roles

What happened so far...

Elicitation of
- Use Cases
- Activities
No/little assignment of responsibilities

Domain model
Components and their Roles

Key Points:

- Mapping between components and use cases/activities
  - Assignment of responsibilities
- Component interaction required to establish use case/activity
  - Interaction Model
- Validation of use cases, activities, domain model
  - Iterative Process

Diagram:

- IVR
  - Login
  - Video Order
  - Retrieval
- User
  - Billing
- Provider
  - ...
Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts
Interaction Diagrams

- Interaction is key ingredient of distributed system's SW architecture:
  - synchronous method-/operation calls
  - asynchronous communication via message exchange
- Coordination/Synchronization via interaction to establish use case
- Communication infrastructure:
  - Object/Component identifiers and interfaces
  - Communication paths (⇒ domain model)
  - Method call / message exchange mechanisms
- Communication defines:
  - Assumptions at environment
  - Commitment towards environment
Interaction Diagrams

UML: Sequence and collaboration diagrams

Sequence Diagrams (SDs):

- Object symbol
- "Lifeline"
- Message
- Deadline

Diagram:

- Client:
  - insertCard
  - enterPIN
  - PIN
  - getBalance
  - 1000

- ATM:
  - <= 10ms

© Ingolf H. Krueger
Interaction Diagrams

Collaboration diagrams:

1. insertCard
2. enterPIN
3. PIN
4. getBalance
5. 1000
Sequence Diagrams

Roots of SDs:

- MSC-96 (Message Sequence Charts, ITU-T Standard Z.120)
  \[\rightarrow\] Specification and documentation of asynchronous telecommunications protocols

- OMSCs (Object Message Sequence Charts, [POSA96])
  \[\rightarrow\] Specification and documentation of control-flow-oriented systems
Sequence Diagrams: Basic Notation

- **External Actor**
- **lck**: Lck
- **ldn**: Ldn
- **rdn**: Rdn
- **Imr**: Imr
- **rmr**: Rmr
- **done**: Done
- **c:Control**
- **l:LM**
- **r:RM**

Time
Sequence Diagrams: Basic Notation

- **object symbol:**
  - Object name
  - Class-/Role-name
  - c:Customer

- **lifeline:**
  - Without activation
  - With activation

- **arrows:**
  - Message name
  - Parameter list
  - transfer(amnt,rcva)
  - neutral
  - synchronous
  - asynchronous
Sequence Diagrams: Activations

Internal Control Flow

Self-call
Sequence Diagrams: Returns

Explicit return arrows indicate return of result and control to caller

Synchronous method call

Explicit return of result and control
Sequence Diagrams: Returns

Implicit return of result and control

Assignment to local variable of object c

status := getStatus()

Return indicator
Sequence Diagrams: Alternatives

Boolean guard: occurrence condition for message

Guard

 alternatives
Sequence Diagrams: Alternatives

Conditional lifelines

c:Customer  b1:Bank  a1:Account  b2:Bank

bal:=getBal()

[bal ≥ a] transfer(rb,ra,a)

[bal < a] deposit(a-bal)

BTransfer(rcvb, rcva, amnt)

add(a-bal)

ack
Sequence Diagrams: Repetition

Enclosure with loop condition

```
b1:Bank
initMultiTransfer()

for all t in transactionSet
    BTransfer(t)
    ack

finishMultiTransfer()
```

Scope of repetition
Sequence Diagrams: Construction and Destruction

Construction

c:Customer → b1:Bank

transfer(rb,ra,a)

Transaction(ra,a) → t:Transaction

BTransfer(t)

delete

Destruction

ack(t)
Sequence Diagrams: Time Bounds

```
for all t in transactionSet

b1:Bank

initMultiTransfer()

BTransfer(t)

ack

finishMultiTransfer()

b2:Bank

Starting point

≤ 10sec

Endpoint
```
Sequence Diagrams: Time Bounds

b1: Bank

initMultiTransfer()

for all t in transactionSet

a

BTransfer(t)

ack

b

Time bound

{b-a ≤ 1 sec.}

finishMultiTransfer()

Time bound

Endpoint
Sequence Diagrams: State Markers

- Main use for SDs: “global” communication patterns
- Every interaction can change component state!
Summary Interaction Diagrams

• UML offers
  – Sequence diagrams
  – Collaboration diagrams

for modeling communication patterns

• SDs support
  – neutral, asynchronous, synchronous communication
  – alternatives, repetition, object creation/deletion, time bounds, state symbols

• High potentials for interface specification
  (structure & behavior)
Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts
From Interaction Models to State Models

- Interaction model:
  => Projection of system behavior onto services/use cases
From Interaction Models to State Models

- State model:
  \[\Rightarrow\) Projection of system behavior onto individual components

![Diagram of state model]

Object 1

Overall system run

idle

locking

unlocking
“A state machine is a localized view of an object, a view that separates it from the rest of the world and examines its behavior in isolation. It is a reductionist view of a system. This is a good way to specify behavior precisely, but often it is not a good way to understand the overall operation of a system.” [RJB99]
From Interaction Models to State Models

State:
Qualitative “snapshot” of an object, defined by

- Assignments to attributes
- Control state (program counter A implicit attribute)
- Defined reactions to incoming messages

Transition:
- State change
- Triggered by event (such as message receipt)
- Guarded by constraint (such as certain data state)
- Defines reactions to incoming messages
From Interaction Models to State Models

History of statecharts

- Background: Theory of finite automata (~ 1950-60)

- Origin (David Harel, 1987):
  - Mealy-/Moore-Automata
  - Hierarchy
  - Concurrency

- Tool support for statecharts available

- Many semantics definitions for statecharts
Example: Controller for Central Locking System
Statename

entry / action1
exit / action2
do / activity
statecharts – Basics

event [precond] / action^event-list

Source

Destination

transition

Source state

Target state
entry- and exit-Actions

entry / a1
exit / a2
do / activity
statecharts – Basics

State activities

Activity executing while this state is active

do / lock doors and trunk

anonymous transition; fires, when “do”-activity over
Hierarchy: OR-states/AND-states

- Problem of “flat” automata
  - No handle at abstraction
  - “Explosion” of graphical notation

- Examples:
Hierarchy: OR-states/AND-states

- Problem of “flat” automata
  - No handle at abstraction
  - “Explosion” of graphical notation

- Examples:

```
empty ──► filled
     
full ──► emptied

heated ──► cooled

cold ──► hot
```

Combined properties
“ideal picture”
Hierarchy: OR-states/AND-states

Solution in UML statecharts:

OR-states:
- Hierarchical
- Every state can contain an entire statechart
- At any time only one “substate” of a “superstate” is active
Hierarchy: OR-states/AND-states

Solution in UML statecharts:

AND-states:
- Hierarchical, concurrent states (Replacement for “product automata”)
- At any time every substate of a superstate is active
- Substates can communicate via synchronous signal exchange
Hierarchy: OR-states/AND-states

Beware: Broadcasting in AND-states

Transition rules for AND-states with broadcast communication can lead to incomprehensible specs!

Better:
explicit message exchange between separate objects via corresponding interfaces
Summary statecharts

- UML offers statecharts for modeling state-based behavior of individual components

- statecharts support
  - Hierarchy: OR states
  - Concurrency: AND states
  - Broadcasting communication model

- Rule of thumb: prefer use of concurrent components over concurrent states