Introduction to Discrete-Event Modeling

Quiz Study Guide — Chapter 4

Manuel D. Rossetti, PhD P.E.

How to Use This Deck

This deck is a **checklist**, not a summary. Every slide is phrased as something you should be able to *state*, *derive*, or *compute* without looking it up.

- **Question formats:** multiple choice, true/false, matching, fill-in-the-blank, multi-slot fill-in-the-blank, and numeric answers.
- Work the **Drills** slides by hand first; the answers are in the appendix at the end.
- The **Common Traps** slides are built from the actual wrong answers — read them last.

i Scope

This is the **event view** of discrete-event modeling: you describe what happens to the *system* as it encounters entities. Chapter 6 covers the process view.

What This Chapter Covers

Concepts

- DEDS vocabulary: event, action, activity, state, entity
- Scheduled versus conditional events
- Time-persistent versus observation-based variables
- Activity diagrams

The KSL

- `Model`, `Executive`, `ModelElement`, `KSLEvent`
- `Response`, `TWResponse`, `Counter`
- `SResource`, `Queue`, `QObject`
- `EventGenerator`, `SingleQStation`

Objectives — Concepts

You should be able to:

- Define a discrete-event dynamic system and say how the clock advances
- Distinguish a **scheduled** event from a **conditional** event
- Distinguish an **action** from an **activity**
- Classify a variable as time-persistent or observation-based, and average it correctly
- Compute a time-average and a utilization from a hand simulation
- Read the symbols on an activity diagram

Objectives — The KSL

You should be able to:

- Name the top-level container, the event controller, and the abstract base class
- Say exactly when `initialize()` is called and what belongs in it
- Choose between `Response`, `TWResponse`, and `Counter` for a given variable
- Relate capacity, number busy, and number available for an `SResource`
- State the four `Queue` disciplines and the default
- Compute the M/M/1 quantities used in the drive-through pharmacy example

Core Vocabulary

Term	Definition
Event	An instant in time at which the system state changes
Action	An **instantaneous** operation associated with an event — takes **zero** simulated time
Activity	An operation that **takes simulated time**, marked by two events
State	An abstraction of attribute values that has duration between two events
Entity	A conceptual thing that flows through the system, potentially using resources

A **DEDS** evolves dynamically through time and its state changes **only at discrete points in time** — the clock does *not* tick in fixed equal-sized steps.

Scheduled versus Conditional Events

- A **scheduled** event's occurrence time can be expressed as a **function of system time**.
- A **conditional** event depends on outcomes that **cannot be predicted in advance**.

The distinction is *not* about determinism versus stochasticity, and it is *not* about whether time is consumed.

Activities of specified duration — getting a haircut, for instance — are formally called **timed (scheduled) activities**.

Two Kinds of Variables

Time-persistent

Values persist over intervals of time: $N(t)$, $Q(t)$, $B(t)$.

Averaged by **time-weighting**:

$$\bar{Q} = \frac{1}{t_n - t_0} \int_{t_0}^{t_n} Q(t)\, dt$$

Collected with `TWResponse`.

Observation-based (tally)

One observation per customer: A_i, S_i, D_i, ST_i, T_i, W_i.

Averaged by a **simple arithmetic mean**.

Collected with `Response`.

In the bank simulation, $N(t) = Q(t) + B(t)$ — number in system equals number in queue plus number busy.

Utilization

For a resource with c units and $B(t)$ busy units, **utilization** is the **time-average of** $B(t)$ **divided by** c — the total time busy divided by the total time the resource *could* be busy.

In the hand bank simulation (7 departed customers over 31 minutes), $\bar{B} \approx 24/31$ is the **proportion of time the teller was busy**.

For an `SResource` with capacity c, number busy $B(t)$, and number available $A(t)$:

$$c = A(t) + B(t), \qquad U(t) = \frac{B(t)}{c}$$

The KSL Architecture

Class	Role
`Model`	Top-level container for all model elements and the experiment parameters
`Executive`	Controls event execution; works with the calendar to fire events **in time order**
`ModelElement`	Abstract base class giving subclasses access to the scheduler and the model
`KSLEvent`	One scheduled event: time, priority, name, and a generic **message**
`EventActionIfc`	Functional interface whose `action()` method holds the logic run when the event fires

Every `ModelElement` is a child of another `ModelElement` or of the `Model` itself, forming a hierarchy with `Model` at the top. **Names must be unique.**

`initialize()`

`initialize()` is called **once at the beginning of each replication**, conceptually at time $t = 0^-$.

- It is **not** the constructor, and it is **not** called once at construction time.
- It is the **recommended place to schedule initial events**.
- The method that places a future event on the calendar is **`schedule`**.

Response Classes

Class	Use
`Response`	Observation-based (tally) statistics on values observed at event times
`TWResponse`	Time-weighted statistics on time-persistent variables
`Counter`	Increment/decrement a variable, **auto-reset to 0** each replication, collect across-replication statistics
`AggregateTWResponse`	A time-weighted response observing other responses and reflecting their sum
`IndicatorResponse`	A `Response` subclass collecting 1.0/0.0 from a boolean predicate on an observed response

Queues, QObjects, and Stations

- `QObject` instances are **transitory**: an inner class of `ModelElement`, not model elements themselves, typically garbage-collected once no longer needed.
- `Queue` supports four disciplines: **FIFO (the default)**, LIFO, random, and ranked. On a **ranked** queue, removal order is governed by the QObject's `priority` property.
- `enqueue` and `removeNext` automatically generate statistics on the number in queue and the time in queue.
- `SingleQStation` combines an `SResource` with a `Queue`. When the resource is seized, the end-of-processing event **attaches the customer as the event's message** so it can be retrieved when the event fires.
- `QObjectReceiverIfc` is the SAM interface promising `receive()`, which stations implement so QObjects can be sent between them.

The Event Generator

The KSL `EventGenerator` is most analogous to the **CREATE module** in other simulation packages — it periodically generates arriving entities or events.

It can be turned off, suspended, and resumed during a replication. **Once turned off within a replication, it cannot be restarted in that same replication.**

The Up-and-Down Component

State is tracked with a `TWResponse`, because it is a time-based variable needing time-weighted statistics.

From renewal theory, with mean up time θ_u and mean down time θ_d:

$$P\{\text{UP}\} = \frac{\theta_u}{\theta_u + \theta_d}, \qquad \text{expected cycle length} = \theta_u + \theta_d$$

With $\theta_u = 1.0$ and $\theta_d = 2.0$: $P\{\text{UP}\} = 1/3$ and the cycle length is 3.0.

Replications and M/M/1

A **replication** is a sample path that starts and ends under the same conditions, so that observations **across** replications are IID.

For the drive-through pharmacy, an M/M/1 with arrival rate λ and service rate μ:

$$\rho = \frac{\lambda}{\mu}, \qquad L_q = \frac{\rho^2}{1-\rho}, \qquad W_q = \frac{L_q}{\lambda}$$

In queueing notation the **M** stands for **Markov**. Two stations in series, each with its own queue and server, form a **tandem** queue.

Activity Diagram Symbols

Symbol	Meaning
Circle labeled “queue”	A waiting line in the system
Rectangle	An activity, with an appropriate label inside
Small circle labeled “resource”	A resource within the system
Dotted line	**Seizing or releasing of a resource**
Zigzag line	Creation or destruction of an entity

Drills

1. Drive-through pharmacy: mean inter-arrival time 6 minutes, mean service time 3 minutes. Compute ρ, L_q, and W_q.
2. Hand bank simulation through time 31 with 7 departed customers and waiting times $0, 0, 2, 5, 5, 5, 7$. Compute the simple average waiting time $\bar{W}$.
3. Up-and-Down Component with mean up time 1.0 and mean down time 2.0. Compute the expected cycle length and $P\{\text{UP}\}$.
4. A teller is busy for a total of 24 minutes out of a 31-minute run with a single teller. What is the utilization?

Common Traps

- The simulation clock does **not** advance in fixed equal ticks.
- Time-persistent and observation-based variables need **different** averaging — time-weighted versus arithmetic.
- `initialize()` is called **each replication** at $t = 0^-$; it is not the constructor.
- The `EventActionIfc` method is **`action()`**, not `execute()` or `run()`.
- Model element names **must** be unique — no automatic suffixing.
- $L_q = \rho^2/(1-\rho)$, **not** $\rho/(1-\rho)$ (that is L for M/M/1).
- A **dotted** line on an activity diagram means seize/release, not entity flow.
- An `EventGenerator` turned off within a replication **cannot** be restarted that replication.

Appendix — Drill Answers

1. $\lambda = 1/6$, $\mu = 1/3$, so $\rho = (1/6)/(1/3) = \mathbf{0.5}$; $L_q = 0.5^2/(1-0.5) = 0.25/0.5 = \mathbf{0.5}$; $W_q = L_q/\lambda = 0.5/(1/6) = \mathbf{3.0}$ minutes.
2. $\bar{W} = (0+0+2+5+5+5+7)/7 = 24/7 = \mathbf{3.4286}$ minutes.
3. Cycle length $= 1.0 + 2.0 = \mathbf{3.0}$; $P\{\text{UP}\} = 1/3 \approx \mathbf{0.3333}$.
4. $\bar{B} = 24/31 = \mathbf{0.774}$ — the proportion of time the teller was busy.