**Pre-midterm Notes:**

1. **Logical time** is fully abstract and virtual representation. It increases or increments with each event or step. E.g.: Logical time steps whenever a new customer arrives in a queueing simulation.
2. **Real time:** is that approximates to physical time in a system, but still is abstracted. It approximates physical time. Rate of change is assumed to be 1. A computer clock that ticks at a defined interval e.g., 1 millisecond.
3. **Physical time:** It is an actual time as measured in the physical world based on predictable phenomena. E.g.: Atomic clock that uses the precise frequency of atoms to keep time.
4. **Superdense time:** Modeling of simultaneity of events. It incorporates time instances and indexes. It is a trajectory that helps to see that a model can have multiple events occurring at the same time.
5. **Superdense time is represented lexicographically**. E.g.: . Basically, it is represented as t1 < t2.

**New Section:**

1. **Software Models:** An abstract representation of the key components, relationships, and logic of the real system.
2. **Programming Language:** Languages e.g.: Java, C#
3. **Simulator:** A software implementation of the model's logic and computations. Allows running simulations over time.
4. **Software Specifications:** Detailed technical design documents that specify the algorithms, data structures, interfaces, and other software details.
5. **DEVS (Discrete Event System Specification)** is a framework for developing modular, hierarchical models. DEVS suite is a set of tools that support DEVS modeling and simulation.
6. **Experimental Frame**: The real-world physical system or environment we want to study through simulation. It defines the context.

**New Section:**

1. **Live Simulation:** Involving real people and real systems.
2. **Virtual Simulation:** Involving real people and virtual systems.
3. **Constructive Simulation:** Virtual people with virtual systems. Large scale simulation but lower realisms.

**New Section:**

1. **Conceptual:** High level of abstraction.
2. **Declarative:** Collection of states and their subsequent changes.
3. **Functional:** Defines system in terms of functions. Outputs are generated based on external input stimuli.
4. **Constraint:** Collection of states and their relationships in a constrained network.
5. **Spatial:** In terms of geometry.

**New Section: Systems Theory**

1. **Observation**: Inputs and Outputs are observed over some period.
2. **I/O Relation**: Pair of I/O matched one-to-one over some period.
3. **I/O Function**: Represents data analytically using mathematical equations rather than just empirically observed data. Each pair of input/output  
   has an associated initial state.
4. **I/O System:** The output can depend upon either state only or on both state and input trajectories.
5. **Coupled System:** output trajectories are generated from interacting I/O systems and lower level coupled systems.

**New Section: Model & Simulation Interoperability, Performance and Composability**

1. **Models and Simulation** are distinct but complementary to each other. Models are conceptual abstraction whereas Simulation execute those abstractions. Together they enable computational studies of systems at any abstraction level.
2. **Modeling and Simulation:** Support creating and assessing real, fictitious hybrid systems.
3. **Experiments and Evaluation: Model verification and Simulation validation** is highly required for user acceptance and adoption.
4. Effective Modeling and Simulation requires a variety of computational platforms, frameworks, tools, and standards.

**New Section: Model & Simulation Interoperability**

1. Interoperability should be generic across platforms. Accommodate wide range of models and simulations from different domains with different designs. Flexibility is a key.
2. Allow hierarchical construction of models and simulations from having different components.
3. Well defined functionality and interfaces standards between components.
4. Support full life cycle needs: Design, Testing and Operational Use.

**New Section: Model & Simulation Composability**

1. It is synthesizing a system’s overall behavior by combining component’s sub-behavior.
2. (Integrated whole as sum of the parts)
3. Enable re-use of models and simulations in a new context through composition.

**New Section: System-Theoretic Modeling Formalisms:**

1. **Difference Equation:** Represent system state transitions in discrete time steps. Model form:. Modeling population growth year to year.
2. **Differential Equation:** Represents continuous system state change over time. Model form:
3. **Discrete Event Equation:** Represent state changes at discrete event occurrences.

Model Form: . Modeling inventory level based on customer orders and shipments.

1. DESS: Differential Equation
2. DEVS: Discrete Event Equation
3. DTSS: Discrete-time Equation
4. Difference Equation
5. Classical and Atomic Model specifications
6. Discrete (-time) Model Systems
7. A diagram of a function

   Description automatically generated with medium confidence

A graph with a line going up

Description automatically generated

1. IOFO:
2. 𝑇𝑟𝑒𝑎𝑑𝑚𝑖𝑙𝑙 = 𝑇, 𝑋, 𝑌, Ω, 𝐹

A white paper with black text

Description automatically generated

A diagram of a graph

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a math equation

Description automatically generated

A white paper with text and numbers

Description automatically generated

A close-up of a line of letters

Description automatically generatedA white text with black text

Description automatically generated

A white paper with black text and numbers

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A white paper with black text and numbers

Description automatically generated

A screenshot of a math test

Description automatically generated

A diagram of a graph

Description automatically generated

A screenshot of a paper

Description automatically generated

A diagram of a function

Description automatically generated with medium confidence

1. Continue(e) : update the sigma.
2. A Simple Processor without Queue has three states namely: Phase, Sigma and Job and parameter is processing\_time.
3. 
4. A screenshot of a computer program

   Description automatically generated
5. A diagram of a diagram

   Description automatically generated
6. Processor with Queue has 4 states namely:

* Phase
* Sigma
* Job
* Queue

1. A diagram of a diagram

   Description automatically generated with medium confidence
2. A screenshot of a computer

   Description automatically generated

**Important Notes Post Mid-term:**

This is explaining the concept of using connectors to couple different input/output (I/O) models to create more complex coupled models in a modular way:

* I/O models refer to models that define the inputs and outputs to a system or component. For example, an I/O model of a simple electrical circuit might specify the voltage and current at its terminals.
* Connectors allow different I/O models to be joined together according to certain rules. This allows complex systems to be built up from simpler subsystems.
* Two I/O models that are "distinct" (have independent definitions) may be coupled together using connectors. For example, the electrical I/O model of a circuit can be coupled with a mechanical I/O model of an attached motor.
* Each I/O model is either "elementary" (a basic, atomic model) or "coupled" (built up from multiple elementary and/or coupled models).
* There must exist a non-empty set of connectors defined to determine exactly how two I/O models will be coupled. For example, connectors may map voltages in one model to forces in another model.

In summary, connectors enable modular composition of different I/O models into more complex coupled models by defining mappings between models under specified rules and interfaces. This allows system architectures to be built up in a structured manner.

This is explaining the different types of constraints that can be imposed on the connectors when coupling multiple input/output (I/O) models into a complex interconnected system.

* Singular - The components are not connected at all. The connector set is empty.
* Conjunctive - Components are connected, connector set is not empty. No restrictions on connections.
* Cascade - Connections are only feedforward, no feedback paths. Avoids loops in flow of signals.
* Essentially cascade - Connections can be rearranged but system behavior stays the same. Connectors give same overall input-output relationship.
* Pure feedback - There exists at least one feedback connector from an output back to an input of the same component. Creates a closed-loop subsystem.
* Mixed - Connectors cannot be classified as any of the other constraint types. Mix of multiple types of connections.

The constraints restrict how components can be interconnected. Different constraints lead to different overall system characteristics in terms of dynamics, stability, etc. Appropriate constraints must be chosen based on desired behavior when coupling models into complex systems.

Here is a basic explanation of how models are coupled in DEVS:

DEVS stands for Discrete Event System Specification. It is a formalism for modeling systems that change state based on discrete events over time.

A basic DEVS model, also called an "atomic model", specifies the behavior of a single component or subsystem. It defines how that component transitions between states and outputs events in response to input events.

A "coupled model" allows connecting multiple DEVS models (whether atomic or other coupled models) to form a larger, more complex model. It defines the connections and coupling between components.

For example, atomic models of an electrical switch, a light bulb, and a battery could be connected in a coupled model to represent a simple handheld flashlight system.

The key idea is that coupled models can themselves be reused as components within even larger coupled models. This means complex systems can be built in a hierarchical, modular manner using layers of coupled models.

For example, the flashlight coupled model could relate to atomic models for human limbs and muscles to study ergonomics. This demonstrates hierarchical composition where coupled models get reused as components in new coupled models.

# There are three types of couplings:

1. External Input Coupling (coupled model input mapped to input of a component inside a coupled model)
2. External Output Coupling (output port of a component in a coupled model is mapped to output port of a coupled model)
3. Internal Coupling (output port of a component in a coupled model is mapped to input port of a component in a coupled model. Basically, couplings of siblings in a model).

# Explanations of an Experimental Frame in a Coupled Model:

The experimental frame is a concept in modeling and simulation that refers to the arrangement of a generator and transducer that facilitates running experiments:

The Generator:

* This is a model that generates input conditions and events to stimulate the system under test. For example, a signal generator that produces test waveforms.

The Transducer:

* This observes and measures the response outputs from the system under test. For example, an oscilloscope that plots measured signals.

Together in an experimental frame, the generator and transducer allow:

1. The generator applies test inputs to stimulate the system under study (applying a range of input conditions).
2. The transducer observes and measures the outputs from the system (capturing the response).
3. By changing generator settings, varied experiments can be run to characterize system behavior under different conditions.
4. The transducer data allows analyzing the I/O relationships and properties of the system model (or physical systems).

So, in summary, connecting complementary generator and transducer models facilitates setting up virtual experiments to stimulate and observe systems under test using modeling & simulation. This leads to understanding of a system by experimentally probing its input-output behavior using the flexibility of simulation.

# Two types of Model types

1. Classic (sequential operation) has atomic and coupled models.
2. Parallel (parallel operation) has atomic and coupled models.

# Elements of a coupled model:

1. Components
2. Interconnection: EIC, EOC and IC

# Parallel Coupled Model Specification:

A screenshot of a computer program

Description automatically generated

A group of text on a white background

Description automatically generated

# No Direct Feedback Loops are allowed:

The statement "no direct feedback loops are allowed" in the context of parallel coupled model specification refers to a constraint on how the component models can be connected when coupling them to form a larger parallel model.

Specifically, it means there cannot be any closed chain of connections that go from an output of one component model back to an input of the same component model.

For example:

Allowed:

* Component A output to Component B input
* Component B output to Component C input
* Component C output to Component A input

NOT Allowed:

* Component A output connected directly back to one of its own inputs

This is because such a direct feedback loop within a single component would form an algebraic loop in the overall system of equations, making it much more difficult to properly simulate and solve.

By disallowing direct feedback cycles within each component, the parallel coupled structure avoids algebraic loops and can be numerically solved more easily.

Indirect feedback cycles may still exist between different components. The key constraint is specifically no direct self-feedback within any single component model.

This simplifies solving while still allowing flexible interconnected parallel composition of component models.

# Couplings of an EF Coupled Model (Experimental Frame)

A screenshot of a computer code

Description automatically generated

# Chapter 5:

Simulation Engine for Atomic and Coupled Simulators:

This paragraph is referring to the core simulation engines that are responsible for executing models defined using the DEVS (Discrete Event System Specification) formalism. There are engines for running two types of models:

1. Atomic models:

* These handle execution of basic, individual component models.
* They progress the state of each atomic model over time according to its defined behavior.
* This includes triggering state transitions and output events at scheduled times.

1. Coupled models:

* These deal with hierarchical coupled models built by connecting multiple sub-models together.
* They coordinate the execution of coupled models by managing the communication between their sub-components.
* This includes routing event messages between internally connected sub-models.
* As well as handling any external input and output events.

In essence, DEVS simulators/engines handle running the actual simulation - they provide the mechanisms to interpret the defined models and advance the state by appropriately triggering events and transitions over time.

Atomic simulators step individual components. Coupled simulators handle message passing and sub-model coordination to simulate the overall behavior emerging from couplings.

Together they allow executing hierarchical DEVS models representing complex systems in a modular fashion. The models define the behavior, the engines actualize it over time.

# Basic Relationship between Model and Simulator:

Models:

* Provide specifications for a system - what components it has and how they are connected.
* Define the state variables and functions that determine the system's behavior over time.
* Describe the dynamics like state transitions, outputs, etc.

Simulators:

* Are computer programs that numerically execute the models over time?
* Follow protocols that step through and update the model state variables.
* Interpret the functions and dynamics defined within models.
* Determine when state events and outputs should occur.
* Advance time and state to simulate the evolution of the modeled system.

In essence:

* Models characterize what a system is and how it behaves.
* Simulators actualize the behavior by computationally following rules for executing the models.

So, models provide the specifications that get interpreted, while simulators provide the mechanisms to simulate the models.

The models characterize the structures and dynamics. The simulators contain procedures for executing the model specifications.

Together they allow studying a system by constructing a computational representation of it and observing its simulated behavior.

# Basics of Simulators:

You have correctly identified some key attributes related to time and state in simulation models:

Time Representation:

* Discrete - Time is quantized into discrete steps/slots. Common in digital simulations.
* Continuous - Time evolves continuously as a real number. Used in analog system simulations.

State Variables:

* Represent current condition of the system being modeled at a point in time.
* Simulation engines update state by computing state transition functions.
* Allows mimicking evolution of real system over time.

Time & State Advance Mechanisms:

* Event Stepped: Time jumps irregularly to next scheduled event time. State updates only at event times. Asynchronous.
* Time Stepped: Time advances regularly in fixed increments. State updates every time step. Synchronous/cyclic.

So, in summary, model specification requires defining:

* Time progress (discrete vs. continuous, stepped vs. event-based)
* State variables to mimic system condition.
* State transition behaviors and events over time

And the simulation engine's role is to numerically solve for and advance time & state according to the model specification.

# Some important concepts:

Simulator:

* Computer program that numerically executes the dynamics and behaviors defined within a model specification to mimic the evolution of the real system over time.

Abstract Simulator:

* Defines procedures and algorithms required to execute a model given in a particular formalism. It specifies WHAT needs to be computed without implementation details.

Atomic Simulator:

* Abstract simulator specialized for executing atomic (individual component) models by stepping their state transitions.

Coupled Simulator:

* Abstract simulator capable of executing coupled (hierarchical) models by coordinating message passing and subsystem interactions.

Simulation Engine:

* Actual software implementation of an abstract simulator that provides the specific mechanisms for HOW model execution and state update computations are carried out.

Types of Simulations:

* Sequential - Single processor executes simulation sequentially.
* Parallel - Multiple processors execute concurrent components for speedup.
* Distributed - Components handled by networked processors for large scale execution.

In summary, abstract simulators characterize WHAT computational protocols are needed to execute different modeling formalisms. Simulation engines realize the protocols by providing mechanisms for HOW to actualize executing the models computationally. And simulations can be implemented via sequential, parallel or distributed processing.

# Algorithmic Steps of Abstract Simulator for Classic Atomic Model

This is describing the algorithm steps followed by the atomic model simulator:

Given: Initial simulation time tL = Time of Last Event σ = Model's sigma (time to next internal transition)

The simulator checks:

1.1) No external event AND (tL + σ > current time)  
THEN: Do nothing, just wait

1.2) No external event AND (current time = tL + σ) THEN: Execute output function  
Execute internal transition function  
(Model state update)

1.3) External event received  
THEN: Execute external transition function  
(Handle/respond to external event)

1. Update tL = current simulation time
2. Schedule next internal event: tN = tL + σ

In essence, the simulator waits until either: A) The next internal transition time is reached OR B) An external event is received.

And then executes the appropriate model functions to either update state or respond to the external event before rescheduling the next internal transition.

This is how the simulator coordinates and executes the dynamics of an atomic model over time.

Very Important Execution Steps in a Coupled Model:

This is outlining the key steps carried out by the coupled model simulator to coordinate execution across multiple coupled component models:

1. Compute Global tN:
   * Get each component's time of next event (tN)
   * Take minimum across all components.
   * This is the global tN
2. Tell all components the global tN
   * If a component has tN == global tN, it is "imminent"
   * Imminent components execute output function to generate new output messages
3. Sort and route the output messages to connected inputs based on the specified couplings.
4. Tell components:
   * If you are the imminent component
   * Or have incoming messages from other components.
   * Or both
   * Then execute your external transition function to update state in response.

The key role of the coupled simulator is to synchronize execution across components by:

* Determining which component(s) have the next earliest event.
* Routing output messages between components appropriately
* Triggering the state updates in components with events

To simulate the coupled model over time.

Final Exam Questions:

Consider a logical-time model that can receive a double number at a time as

input. It does some calculations on any received input. Suppose when the model receives an

input value 7.0, after 1/3 unit of time, it outputs the received input. Then, the value SEVEN is

immediately outputted (i.e., the simulation time is not increased). Assume there is no input at

the time an output is produced.

a) [2 points] Identify what method or methods can be used to develop and simulate the

above model.

Ans: Based on the description of the logical-time model, here are some methods that can be used to develop and simulate it:

a) Methods to develop the model:

* Discrete event modeling: The model behavior is defined based on discrete events that occur at moments in time. This fits the description of receiving an input, waiting for a specified delay, and producing outputs.

I would recommend using a discrete event simulation approach to model and simulate this system. This method is applicable since the model behavior involves distinct events that occur at specific points in time:

The model can be defined as having the following key events:

* input\_arrival - A double input value arrives, stored in the variable input\_value

Sure, if we model this system using DEVS (Discrete Event System Specification), we can specify external and internal transition functions.

The DEVS model would be defined as:

Inputs: double Outputs: double, string

States:

* WAITING: waiting for input
* DELAY: delaying for 1/3-time units after input received

External Transition Function: δext (s, e, x)

* s = current state
* e = elapsed time since last transition
* x = value of the arrived input

δext(WAITING, \_, x):  
state.input\_value = x  
return DELAY, 0

δext (DELAY, TE, \_):  
return DELAY, TE (elapsed time since last transition)

Internal Transition Function:  
δint (s, e)

* s = current state
* e = time elapsed since last transition

δint(DELAY, TE): if TE >= 1/3  
output state.input\_value  
output "SEVEN" return WAITING, 0 else return DELAY, 0

So, in summary, the external transition function handles arrival of an input value, records it, and transitions the state to start the delay. The internal transition function handles the expiration of the delay, generates the required outputs, and resets the state.

Consider the abstract parallel DEVS atomic simulator. Identify what kinds of

concurrency are supported by this abstract protocol.

The abstract parallel DEVS (Discrete Event System Specification) atomic simulator supports the following kinds of concurrency:

1. Input concurrency: The abstract simulator allows multiple input events to arrive concurrently and be processed in parallel up to the next simulation time. This includes handling simultaneous events targeted at the same or different models.
2. Transition concurrency: The external transition function (δext) and output function (λ) can be executed concurrently for different models that have simultaneous events. This allows models to independently process input events and generate outputs in parallel.

To summarize, the DEVS abstract parallel simulator exploits concurrency across models for processing simultaneous events and collecting outputs. But each DEVS model itself behaves in a sequential manner for deterministic processing of events and state updates. The synchronization is managed by the simulator protocol advancing time.