West Virginia University

PART II – SPE Models

Software Execution Models

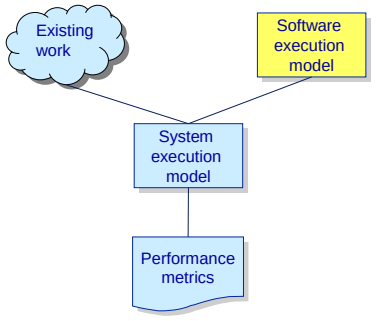
West Virginia University

Outline

- Purpose and properties of software execution models (SEM)
- Execution graph representation
- Solving software execution models
- Case study

West Virginia University

SPE Models



Purpose of Software Execution Models

- Early modeling is essential to ensure that the software architecture will meet performance objectives
- Problem - Early in the development process we do not have sufficient knowledge to model performance precisely
- Solution - Construct the simplest possible models that capture essential performance characteristics

Properties of Software Execution Models

- Software execution model characterizes resource requirements of the proposed software alone, in the absence of
  - other workloads
  - multiple users
  - delays due to contention for resources
- It provides analysis of
  - best-case response times
  - worst-case response times
  - average response times

Properties of Software Execution Models

- Software execution models can identify serious performance problems at early design phases
  - If the predicted performance is unsatisfactory there is no need to build system execution model
- The absence of problems in software execution model does not mean that there are no problems
  - Contention for system resources could cause problems
    - These problems may be corrected with
      - Software design alternatives
      - Hardware configuration alternatives

Outline

- Purpose and properties of software execution models (SEM)
- Execution graph representation
- Solving software execution models
- Case study

Execution graph representation

- SEM are represented with execution graphs
  - For each key performance scenario we construct execution graph
- Execution graphs are based on elementary graph theory
  - Nodes represent processing steps (program statements that perform a function)
  - Arcs represent the order of execution
- Execution graphs are similar to program flowcharts but not the same; they show
  - only those paths that are key to performance
  - frequency of path execution

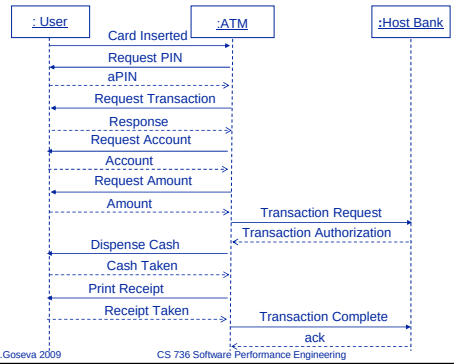
Execution graph representation - contd

- It is possible to construct different execution graphs that represent the same software
- These graphs may differ in representation of
  - Software hierarchy
  - Details of abstraction
- There is no single “right way”

Basic execution graph notation

Basic node		Functional component at the current level of detail
Expanded node		Function will be refined; details will be given in an associated subgraph
Repetition node		Subsequent nodes are repeated n times; the last node in the loop has an arc to this node
Case node		Attached nodes are conditionally executed; Each has an execution probability
Pardo node		Attached nodes are executed in parallel; all must complete before proceeding
Split node		Attached nodes represent new processing threads; They need not all complete before proceeding

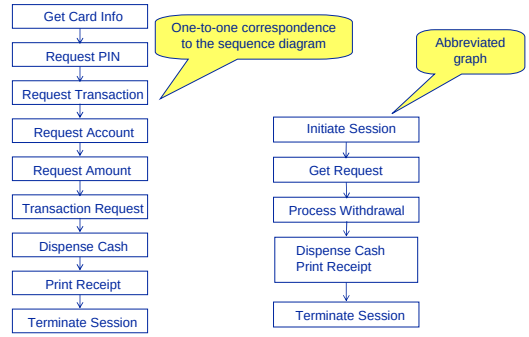
Sequence diagrams for ATM Withdrawal



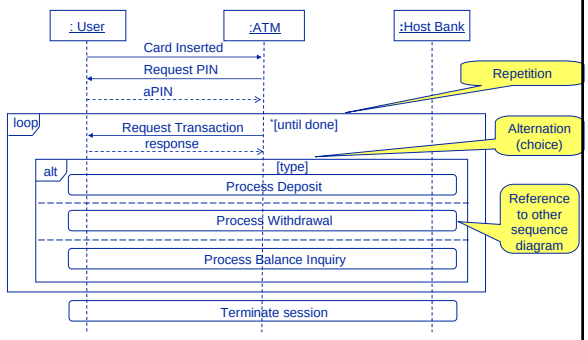
Sequence diagrams for ATM Withdrawal - contd

- For single-threaded scenarios or scenarios with sequential flow of control transforming sequence diagram to execution graph is straightforward
- Each message received by an object triggers an action – follow the message arrows through the sequence diagram and make each action a basic node in the execution graph
- In many cases, individual actions are not interesting from the performance prospective, so several of them may be combined together in a single basic node

Execution graph for ATM withdrawal



General ATM sequence diagram



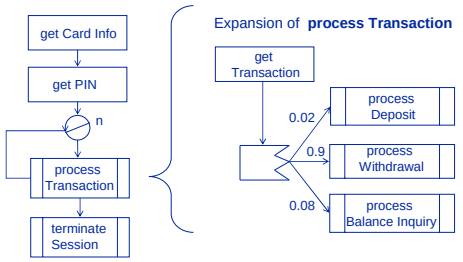
General ATM sequence diagram - contd

- If UML with MSC extension is used (UML 2.0 or later) repetition and case nodes are easy to identify
- If not
  - to identify repetitions - walkthrough the sequence diagram
  - to identify alternative steps (case node) – look at different sequence diagrams that represent scenarios from the same use case

General ATM sequence diagram - contd

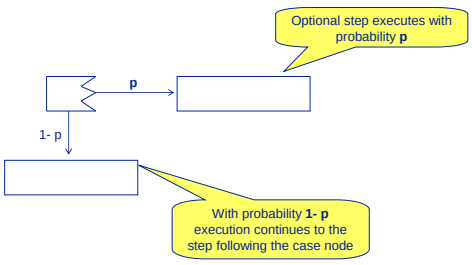
- Reference is most easily represented as an expanded node
- For scenarios that involve multiple threads of control or distributed objects more effort is needed to account for communication and synchronization delays

Execution graph for general ATM sequence diagram

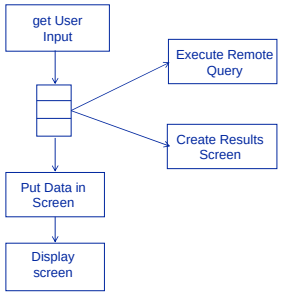


Case node for optional step

- Probabilities of conditional paths on the case node need not sum to 1

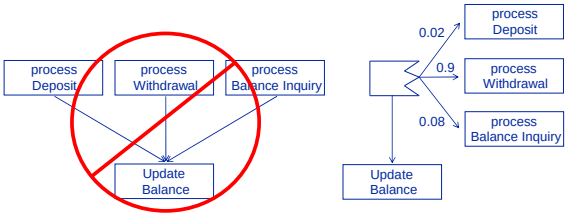


Example for Pardo (parallel do) node



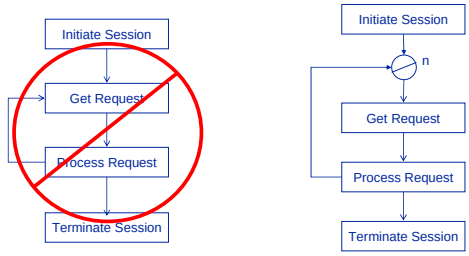
Execution graph restrictions

- Initial node restriction: graphs and subgraphs can have only one initial node



Execution graph restrictions - contd

- Loop restriction: All loops in the graph must be repetition loops



Execution graph restrictions - contd

- Restrictions do not restrict the modeling power of the execution graphs – for each graph that violates restrictions there is an equivalent legal representation
- Restrictions simplify the solution algorithms and enable quick solution of many alternatives

Outline

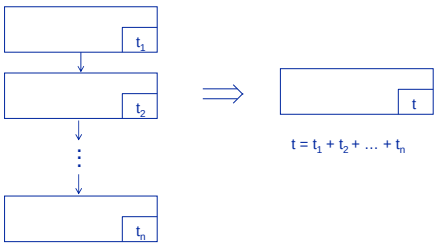
- Purpose and properties of software execution models (SEM)
- Execution graph representation
- Solving software execution models
- Case study

Solving SEM – reduction rules

- Graph reduction method
  - Identify basic structure
  - Compute the time for the structure
  - Replace the structure with a single node whose "time" is the computed time
- Basic structures
  - Sequences
  - Loops
  - Cases

Sequential structure

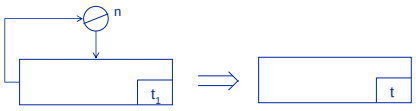
- Sequential structure – sum of the times of the nodes in sequence



$t = t_1 + t_2 + \ldots + t_n$

Loop structure

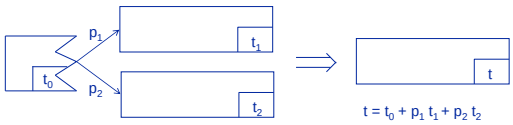
- Loop structure – multiply node time with the loop repetition factor



$t = n\, t_1$

Case structure

- Computation differs for the shortest path, longest path, and average analysis
  - Shortest path – minimum of the times for the conditionally executed nodes
  - Longest path – maximum of the times for the conditionally executed nodes
  - Average analysis – sum of each node's time multiplied by its execution probability

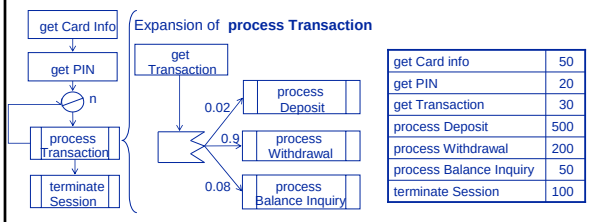


$t = t_0 + p_1\, t_1 + p_2\, t_2$

Parallel structure

- Best case – use the longest of the concurrent paths (other parallel paths are complete when the longest concurrent path completes)
- Worst case – sum of the concurrent paths (serialize the parallel paths, i.e., when one path completes the next begins)

Illustration for reduction rules - ATM example



get Card info	50
get PIN	20
get Transaction	30
process Deposit	500
process Withdrawal	200
process Balance Inquiry	50
terminate Session	100

Illustration for reduction rules - ATM example

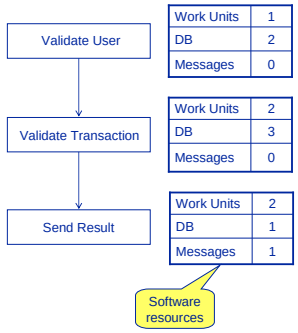
Process Transaction:

Shortest path = get Transaction + ALT + process Balance Inquiry = 30 + 50 = 80
Longest path = get Transaction + ALT + process Deposit = 30 + 500 = 530
Average path = 30 + 0.02*500 + 0.9*200 + 0.08*50 = 224

Session:

Shortest path = get Card Info + get PIN + n*process Transaction-SP + terminate Session
= 50 + 20 + 2*80 + 100 = 330
Longest path = get Card Info + get PIN + n*process Transaction-LP + terminate Session
= 50 + 20 + 2*530 + 100 = 1,230
Average path = get Card Info + get PIN + n*process Transaction-AV + terminate Session
= 50 + 20 + 2*224 + 100 = 618

Solving SEM – example **authorize Transaction**



Work Units specify relative CPU consumption

Range of values from 1 to 5
- 1 - simple task
- 5 - the most complex task

The processing overhead then specifies an approximate number of machine instructions for the simple task

Solving SEM – example **authorize Transaction**

Devices	CPU	Disk	Network
Quantity	1	1	1
Service Units	KInstr.	Phys. I/O	Msgs.

Name, quantities, and service units of the computer devices

Work Unit	20	0	0
DB	500	2	0
Massages	10	2	1

Connection between software resources and computer device usage; for example DB requires 500K CPU instructions, 2 I/Os, and 0 network messages

Service Time	0.00001	0.02	0.01

Service time

Processing step	CPU Kinstr	Phys. I/O	Network msgs
Validate User	1,020	4	0
Validate Transaction	1,540	6	0
Send Result	550	4	1
Total	3,110	14	1

Step 1: Estimate total computer resources for each node

Step 2: Estimate total computer resources using reduction rules

Solving SEM – example **authorize Transaction**

- **Step 3:** Estimate elapsed time by multiplying total resource requirements for each computer resource by the service time for that resource (from the last row), and summing the result for each resource

  3,110 * 0.00001 + 14 * 0.02 + 1 * 0.01 = 0.3211 *sec*

  This is the optimistic estimate of the elapsed time because it excludes queuing delays when multiple processes want to use the same computer resources in the same time

Outline

- Purpose and properties of software execution models (SEM)
- Execution graph representation
- Solving software execution models
- Case study

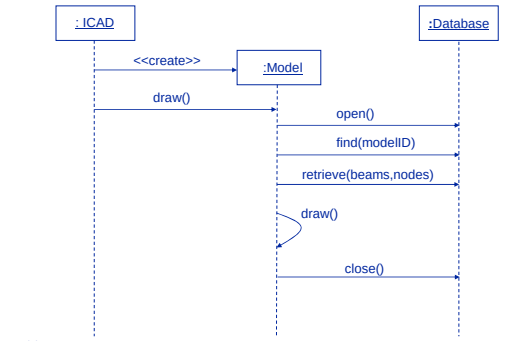
Case study - ICAD

- Interactive computer-aided design (ICAD) application used to
  - draw the model structures such as aircraft wings
  - store a model in a database (several versions of the model may exist within the database)
  - interactively assess the design's correctness, feasibility, and suitability
- ICAD drawing consists of
  - Nodes – position (x,y,z) and additional information
  - Elements
    - Beams – connect two nodes
    - Triangles – connect three nodes
    - Plates – connect four or more nodes

ICAD use cases

- ICAD use cases
  - **Draw** – draw a model
  - **Solve** – solve a model
- We will focus on **Draw** use case and its scenario **DrawMod**
  - Typical model consists of nodes and 2,000 beams (no triangles or plates)
  - Performance objective is to draw a typical model in 10 seconds or less

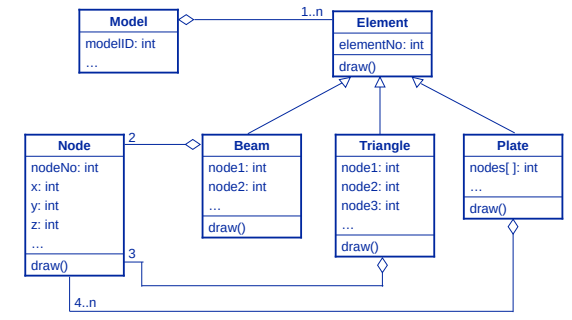
High-level sequence diagram of **DrawMod** scenario



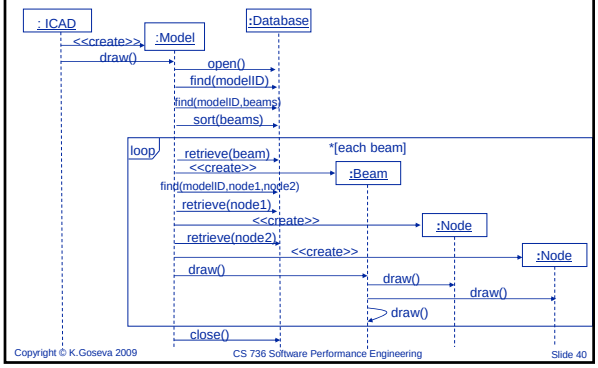
Architecture 1

- Use an object to represent each node and element
  - Flexible
  - All types of elements can be treated in a uniform way
  - New types of elements can be added without changing any other aspect of the application

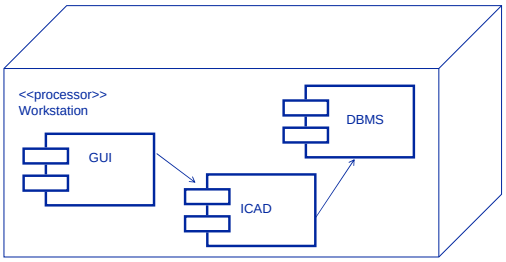
Architecture 1 – class diagram



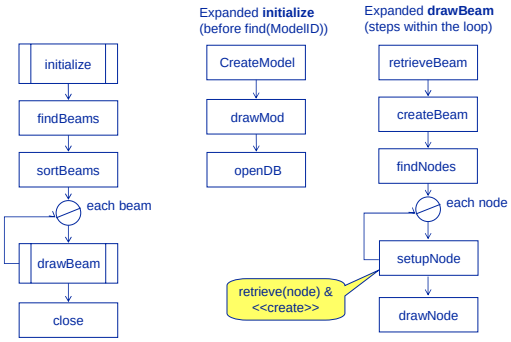
Expanded sequence diagram of **DrawMod** scenario



ICAD deployment diagram



DrowMod execution graph



Specify software resource requirements

- Software resources for ICAD example
  - EDMS – number of calls to the DB process
  - CPU – estimate of the number of instructions executed
  - I/O – number of disk accesses to obtain data from DB
  - Get/Free – number of calls to the memory management operations
  - Screen – number of times graphics operations "draw" to the screen
- We need
  - values for software resource requirements for each processing step in the software execution model
  - number of loop repetitions
  - probability for each case alternative (does not exist in this example)

Values for **DrawMod** software resource requirements

Processing steps	EDMS	CPU	I/O	Get/ Free	Screen
createModel	0	2	0	0	0
drawModel	0	1	3	2	2
openDB	1	2.3	6	1	0
findBeams	1	346	7.08	0	0
sortBeams	1	339	42.28	2	0
close	1	1.5	2	1	0
retrieveBeam	1	2	4.03	0	0
createBeam	0	2	0	0	0
findNodes	1	4.5	4.1	0	0
setupNode	1	4	4.02	0	0
drawNode	0	0.55	0	0	1

* One table for software resource requirements is given instead of separate tables associated with each step

Values for **DrawMod** computer resource requirements

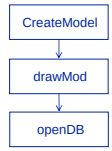
Devices	CPU	Disk	Display
Quantity	1	2	1
Service Units	K instr.	Phys. I/O	Units

EDMS	0.253	0.002	
CPU	1		
I/O	0.1	1	
Get/Free	0.1		
Screen	0.05		1

Service time	0.000005	0.03	0.001

Time for **initialize** step

Expanded **initialize**
(before find(ModelID))



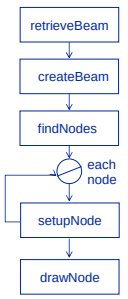
Processing step	CPU Kinstr	Disk Phys.I/O	Display Units
CreateModel	2.000	0.000	0.000
drawMod	1.600	3.000	2.000
openDB	3.253	6.002	0.000
Total	6.853	9.002	2.000

Service time	0.000005	0.03	0.001

Time	3.4265e-5	0.27	0.002

Time for **drawBeam** step

Expanded **drawBeam**
(steps within the loop)



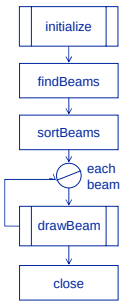
Processing step	CPU Kinstr	Disk Phys.I/O	Display Units
retrieveBeam	2.656	4.032	0
createBeam	2.000	0.000	0
findNodes	5.163	4.102	0
setupNode	4.655	4.022	0
drawNode	0.600	0.000	1
Total	19.729	16.178	1

X 2

Service time	0.000005	0.03	0.001

Time	9.8645e-5	0.48534	0.001

Elapsed time for **DrawMod -** Architecture 1



Processing step	CPU Kinstr	Disk Phys.I/O	Display Units
Initialize	6.853	9.002	2
findBeam	346.961	7.082	0
sortBeams	343.681	42.282	0
drawBeam	19.729	16.178	1
close	2.153	2.002	0
Total	40157.648	32416.368	2002

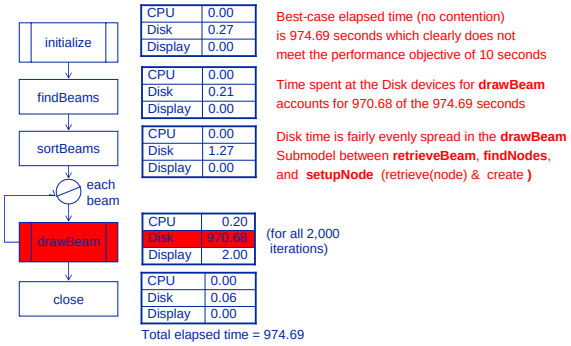
X 2000

Service time	0.000005	0.03	0.001

Time	0.200788	972.491	2.002

Total elapsed time = 974.69

Elapsed time for **DrawMod** - Architecture 1



(for all 2,000 iterations)

Total elapsed time = 974.69

Best-case elapsed time (no contention) is 974.69 seconds which clearly does not meet the performance objective of 10 seconds

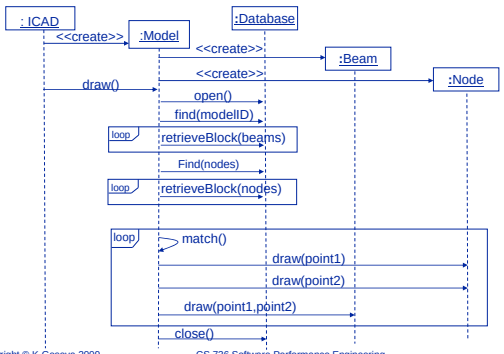
Time spent at the Disk devices for **drawBeam** accounts for 970.68 of the 974.69 seconds

Disk time is fairly evenly spread in the **drawBeam** Submodel between **retrieveBeam**, **findNodes**, and **setupNode** (retrieve(node) & create **)**

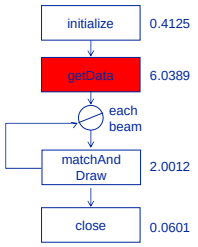
Architecture 2

- New operation to retrieve a block of data - **retrieveBlock()**
  - Retrieve blocks containing 20K of data at a time instead of retrieving individual nodes and beams
  - Single **RetrieveBlock()** can fetch 64 beams or 170 nodes

Sequence diagram of **DrawMod** scenario – Architecture 2



Execution graph and results for Architecture 2



Total elapsed time = 8.5126

Modeling Hints

- Combine related steps that do not have significant effect on performance
  - SEM is an abstraction which includes only details that are relevant to performance
- Use hierarchy
  - Models are easier to understand and modify
  - Expand nodes as your knowledge of the software increases

Modeling Hints - contd

- Use best-case and worst-case estimates for resource requirements
  - If the best-case results indicate that there is a problem, fix it before proceeding
  - If the worst-case results indicate that there is no problem, proceed
  - If there is a problem, look at the processing steps that consume the most resources

West Virginia University

Modeling Hints - contd

- Study the sensitivity of the performance results to the input parameters; identify critical resources and components whose use of these resources should be monitored
- Sensitivity may due to
  - Processing in loops
  - Significant synchronization and resource-sharing delays