

Performance Analysis with Use Case Maps

Murray Woodside

Department of Systems and Computer Engineering

Carleton University

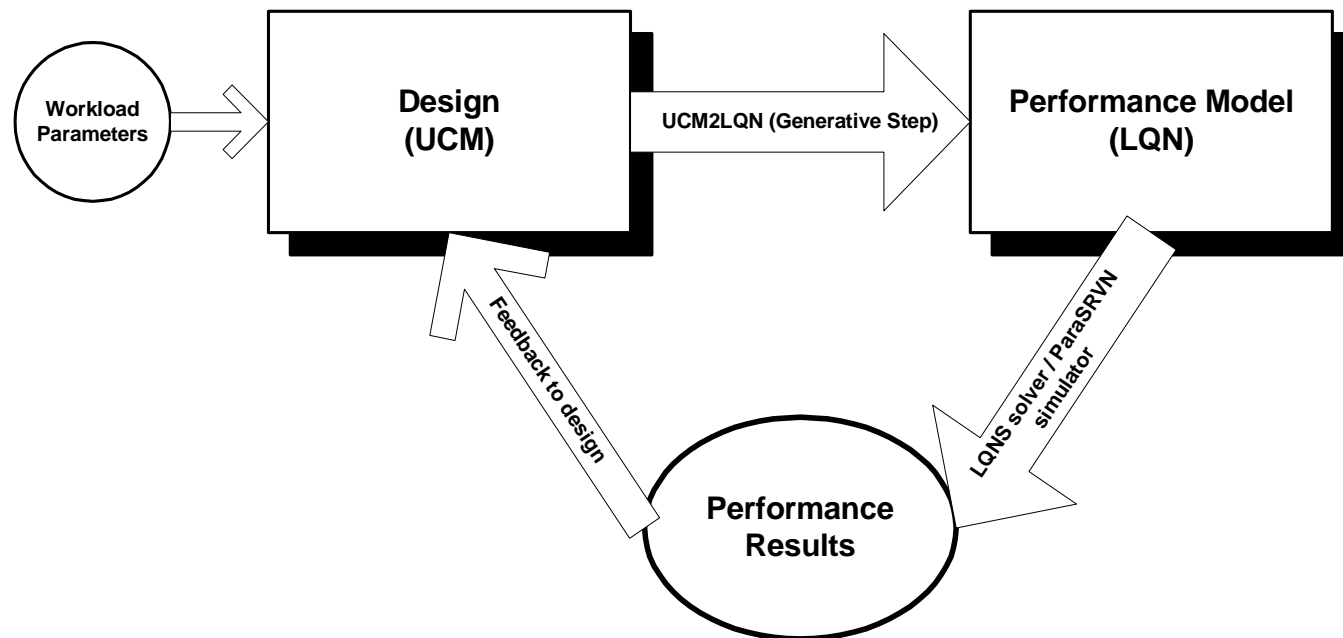
cmw@sce.carleton.ca



Carleton
UNIVERSITY

Early Performance Aware Development (E-PAD)

- Very early analysis of performance
 - Use Case Map (UCM) design specification
 - automatically generated Layered Queuing Network (LQN) performance model
 - workload parameters are budgets, estimates, or values from existing components.

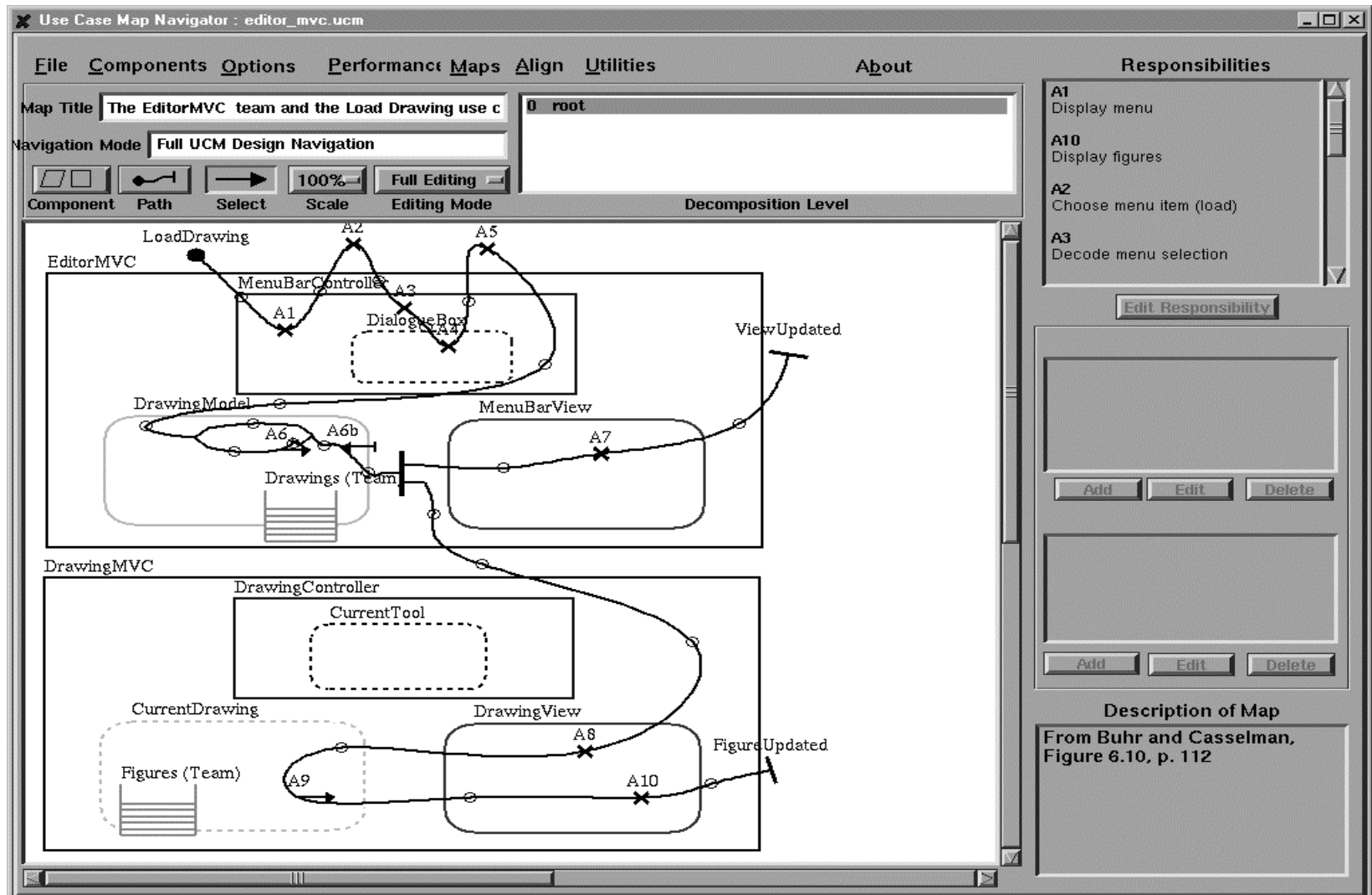


Models

- Use Case Maps (UCMs) represent scenarios executing across a system
 - show the emerging behaviour of the system
 - the basic UCM constructs are *paths*, *components*, and *responsibilities*
 - use them to reason about design and performance
- Layered Queuing Networks (LQNs) specify performance models over a system architecture
 - generate automatically
 - architecture based for traceability
 - the basic LQN constructs are *devices*, *tasks*, *entries*, and *activities*
 - *activities* have parameters for CPU workload and requests to other entries

Begin: Specifications in UCMs

... the UCM Navigator (UCMNav)

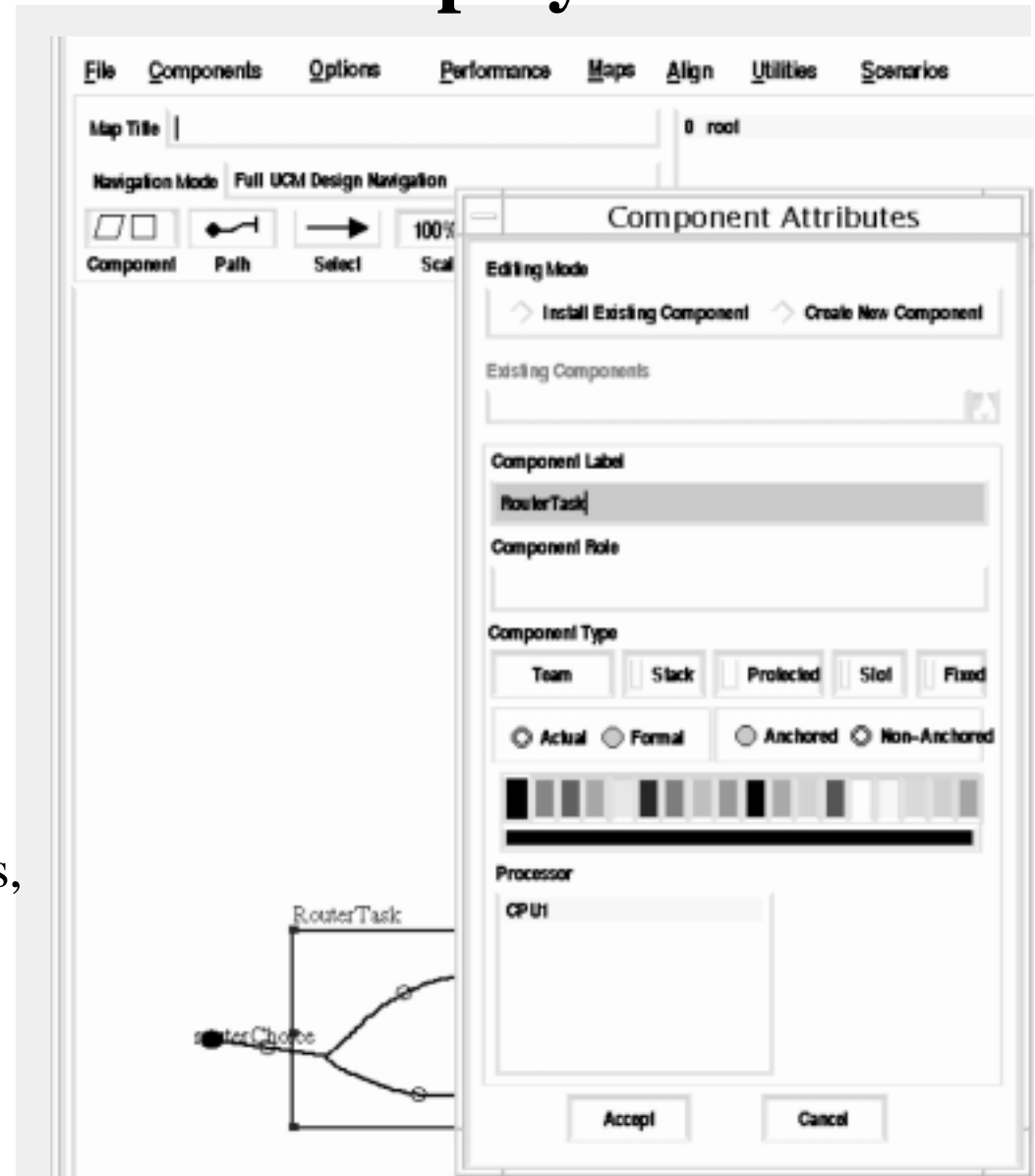


Performance annotations for UCMs

- Deployment
 - declare a list of processors with speed factors
 - bind a component to a processor from a list
 - declare other devices and services
- Path progression parameters
 - probability of a branch, or a dynamic stub
 - loop count
- Operation costs (workloads)
 - annotation attached to each responsibility
 - mean CPU load in time units
 - mean operations on other devices, in operations (disk, attached processor, external service outside the UCM)
- Workload as a mixture of scenarios
 - weighted combination.
- Time points and intervals
 - required delay between time points

Performance annotations: deployment

- elements in a component are bound to it
- component attributes
 - specifies a processor
 - has a “performance -> devices” window to define
 - processors
 - disks
 - attached processors
 - external services (e.g. directories, web servers, data bases)
- device has an operation time



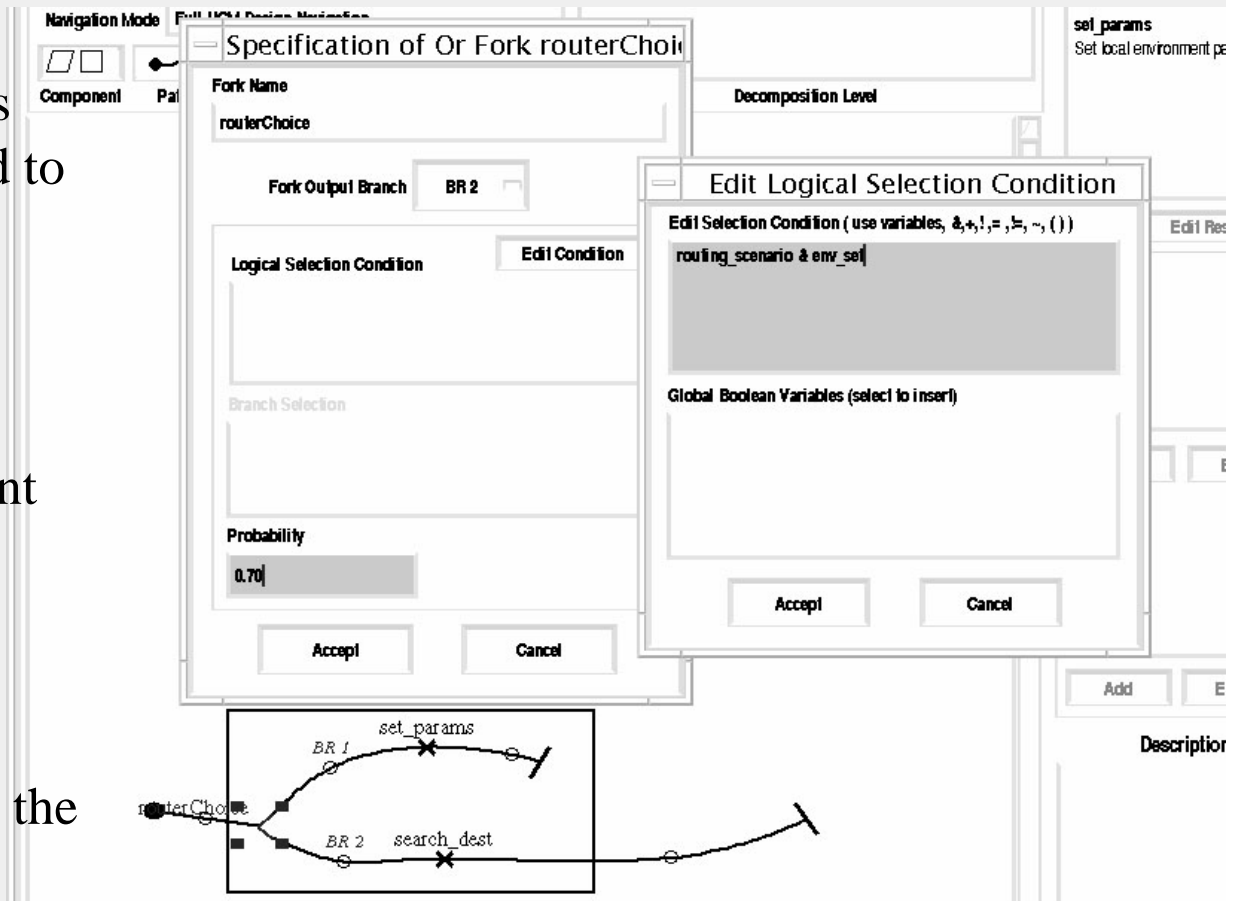
Performance annotations: branch points

- annotations are integrated into other specification windows
 - e.g.: a branch point (OR-fork) has a probability for each branch as well as a predicate for each branch.

.....in performance analysis the probability can be used to specify how often the predicate will be true.

..... here, the upper branch handles setting environment parameters on the router component... it has a probability of 0.30

..... the lower branch does the routing, probability 0.70

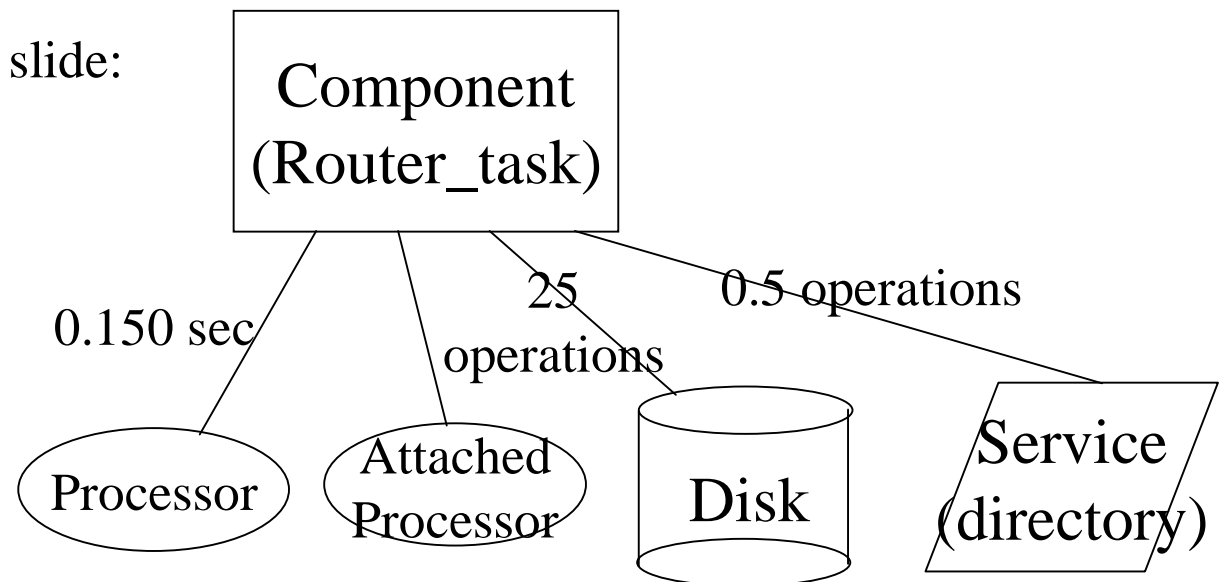


Performance Annotations: Responsibilities

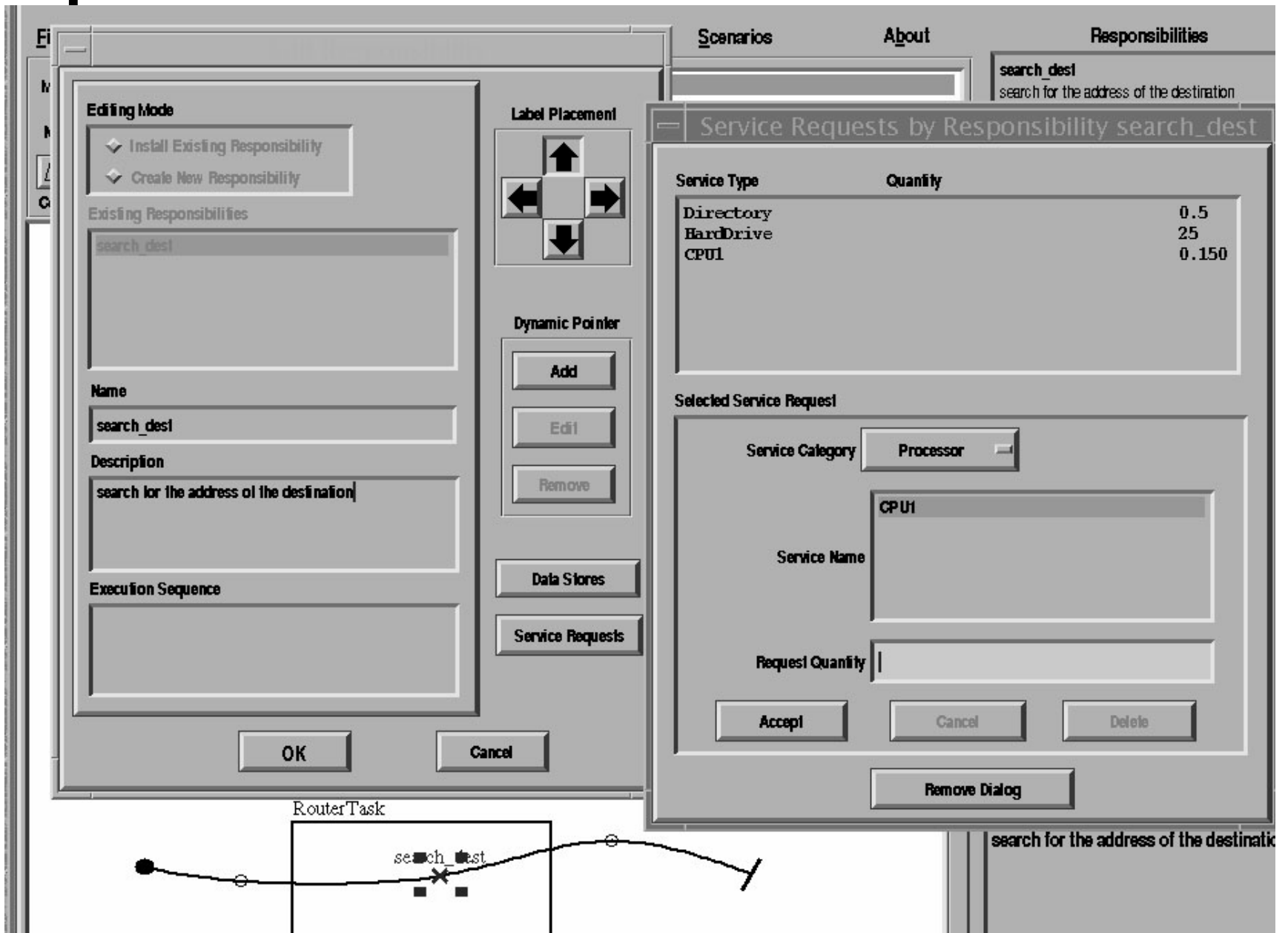
- capture the workload needed to satisfy the responsibility
 - local in this component: CPU, I/O
 - in other components: service requests

Two sub-windows on next slide:

- Responsibility window
- Workload requests sub-window:
 - ... CPU
 - ... disk
 - ... external directoryneeded for 50% of operations



Responsibilities... windows



The screenshot displays two overlapping windows from a software application. The background window is titled 'Responsibilities' and contains the following elements:

- Editing Mode:** Two options: 'Install Existing Responsibility' and 'Create New Responsibility'.
- Existing Responsibilities:** A list box containing 'search_dest'.
- Name:** A text field containing 'search_dest'.
- Description:** A text field containing 'search for the address of the destination'.
- Execution Sequence:** An empty list box.
- Label Placement:** A set of four directional arrows (up, down, left, right) around a central square.
- Dynamic Pointer:** Three buttons: 'Add', 'Edit', and 'Remove'.
- Data Stores:** A button labeled 'Data Stores'.
- Service Requests:** A button labeled 'Service Requests'.
- Buttons:** 'OK' and 'Cancel' at the bottom.

The foreground window is titled 'Service Requests by Responsibility search_dest' and contains:

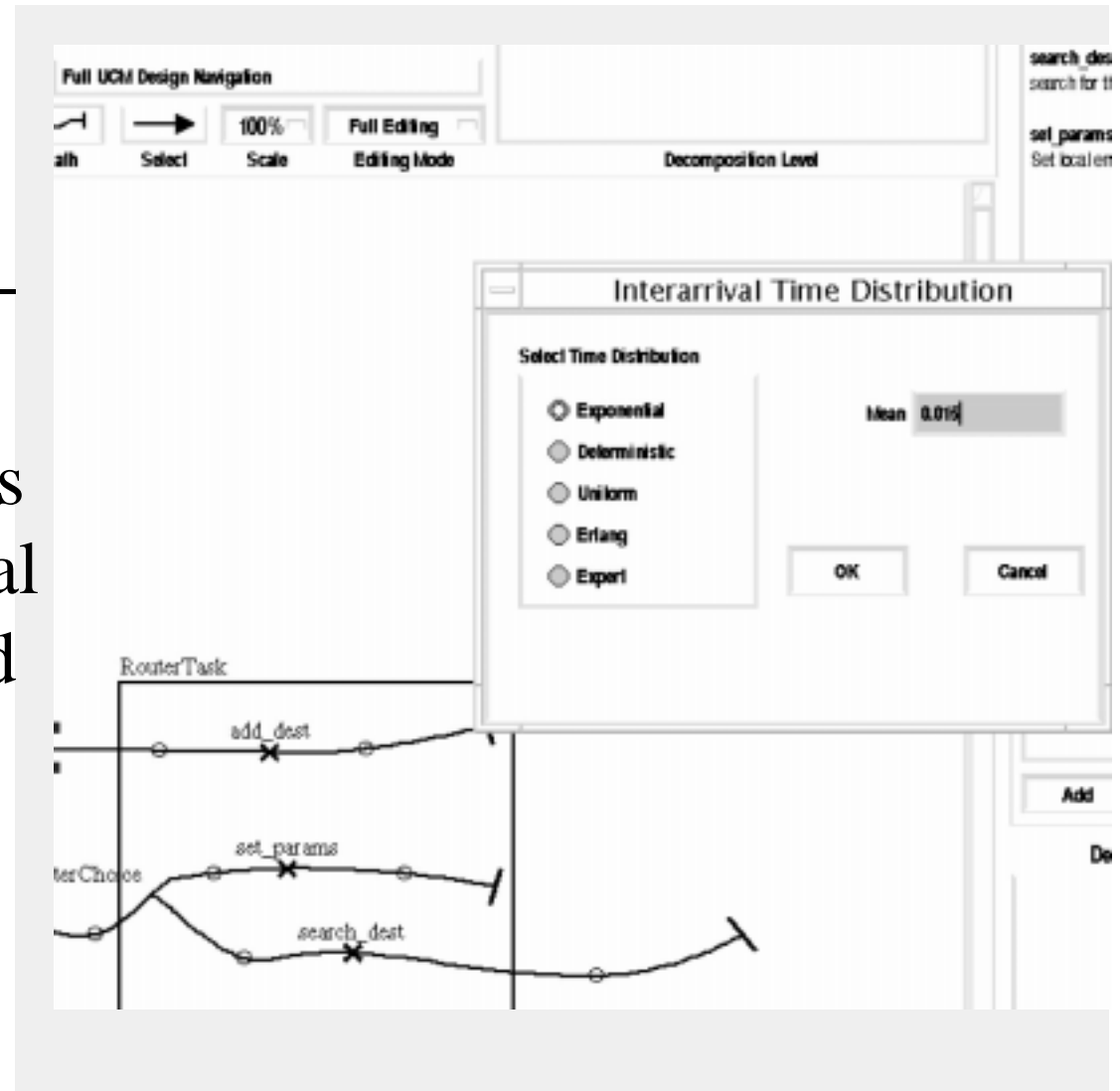
- Table:**

Service Type	Quantity
Directory	0.5
HardDrive	25
CPU1	0.150
- Selected Service Request:**
 - Service Category:** A dropdown menu showing 'Processor'.
 - Service Name:** A text field containing 'CPU1'.
 - Request Quantity:** An empty text field.
 - Buttons:** 'Accept', 'Cancel', and 'Delete'.
 - Remove Dialog:** A button at the bottom.

At the bottom of the image, there is a diagram showing a curved line with a dot at the start and an arrow at the end. The line passes through a box labeled 'RouterTask' and a point labeled 'search_dest'.

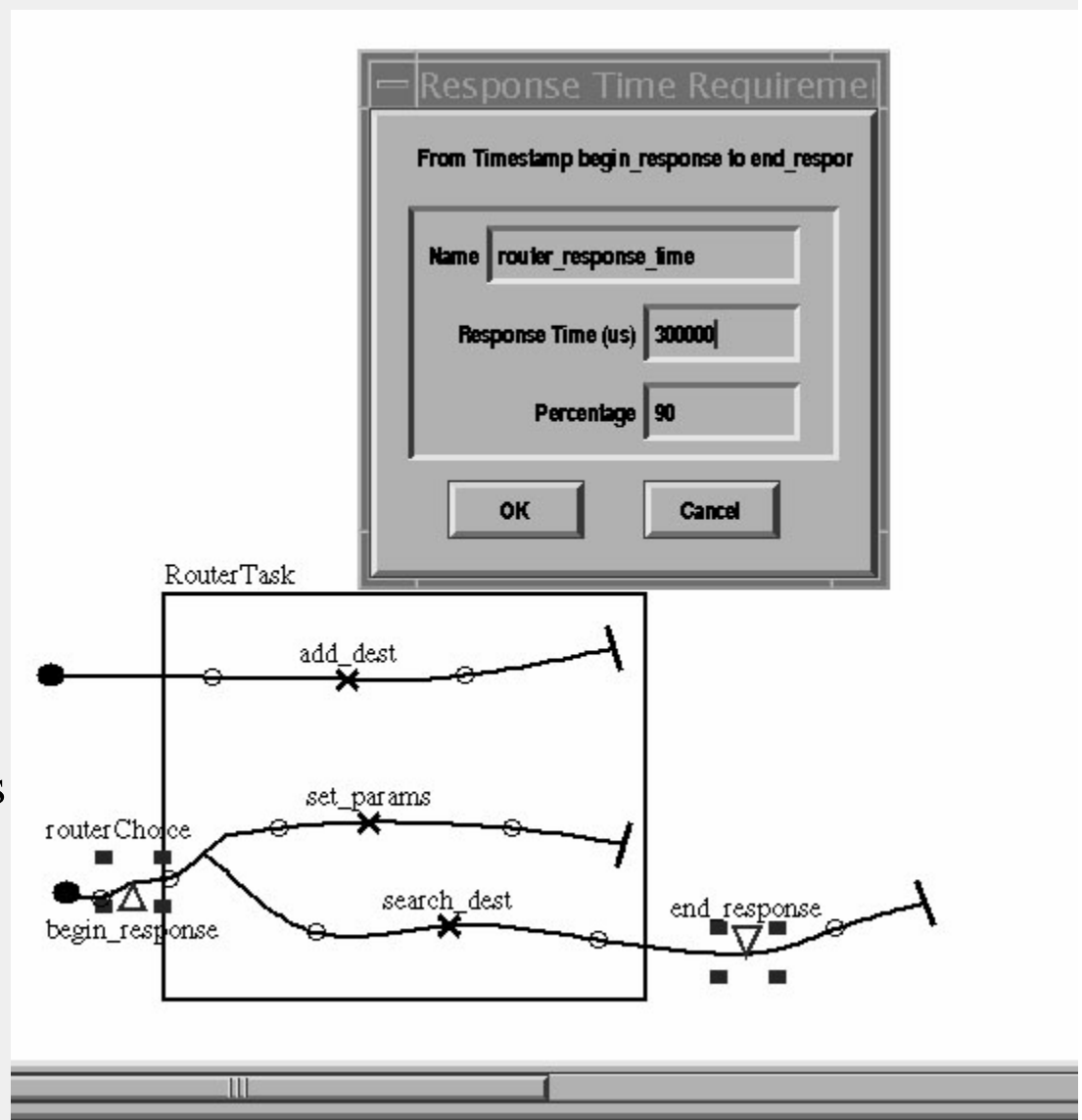
Workload intensity given by scenario-start annotation

- Arrivals of scenario initiation events (*open load*) by inter-arrival times
- *Closed* workload has a number of potential sources which spend time outside the system, and then arrive (planned).



Time points and delay specifications

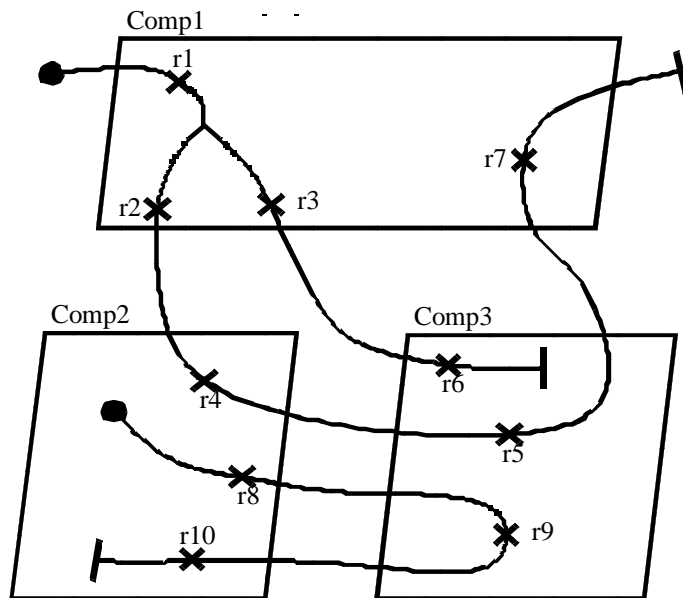
- timestamp points (arrowheads)
- for a pair of these points, a delay specification and an optional percentage
 - percentage specifies a percentile requirement
 - 100% is a hard requirement
 - if empty, then the spec is for the mean
- documentation function, some use in modeling



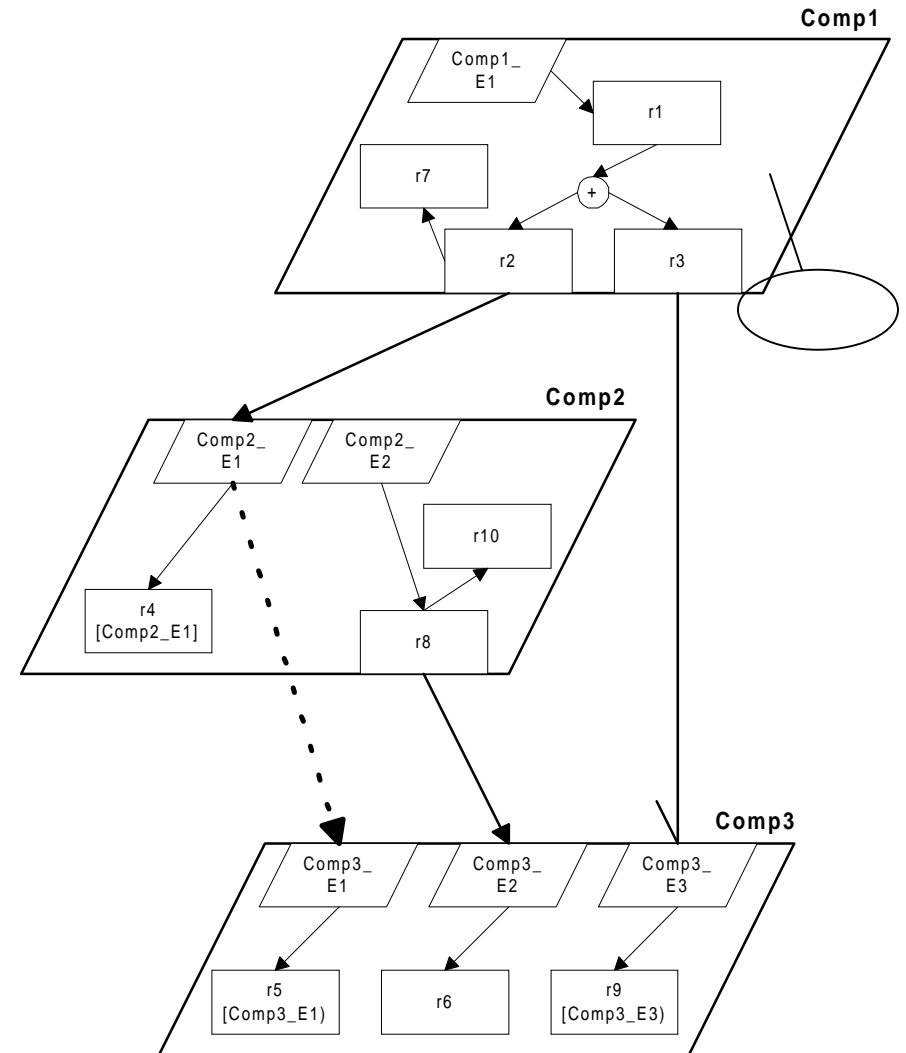
UCM2LQN - UCM to LQN Generation

- traverse UCM paths one at a time
 - point-by-point traversal
- identify components and path constructs
- identify interactions between components
 - identify the types of interactions based on the way in which paths traverse components
 - synchronous, with return
 - asynchronous, no return
 - forwarding.... return at end of longer path
- create LQN objects
 - incorporate UCM workload parameters, with defaults if they are missing
- output LQN model

Relationships between the models



*paths, components,
and responsibilities*

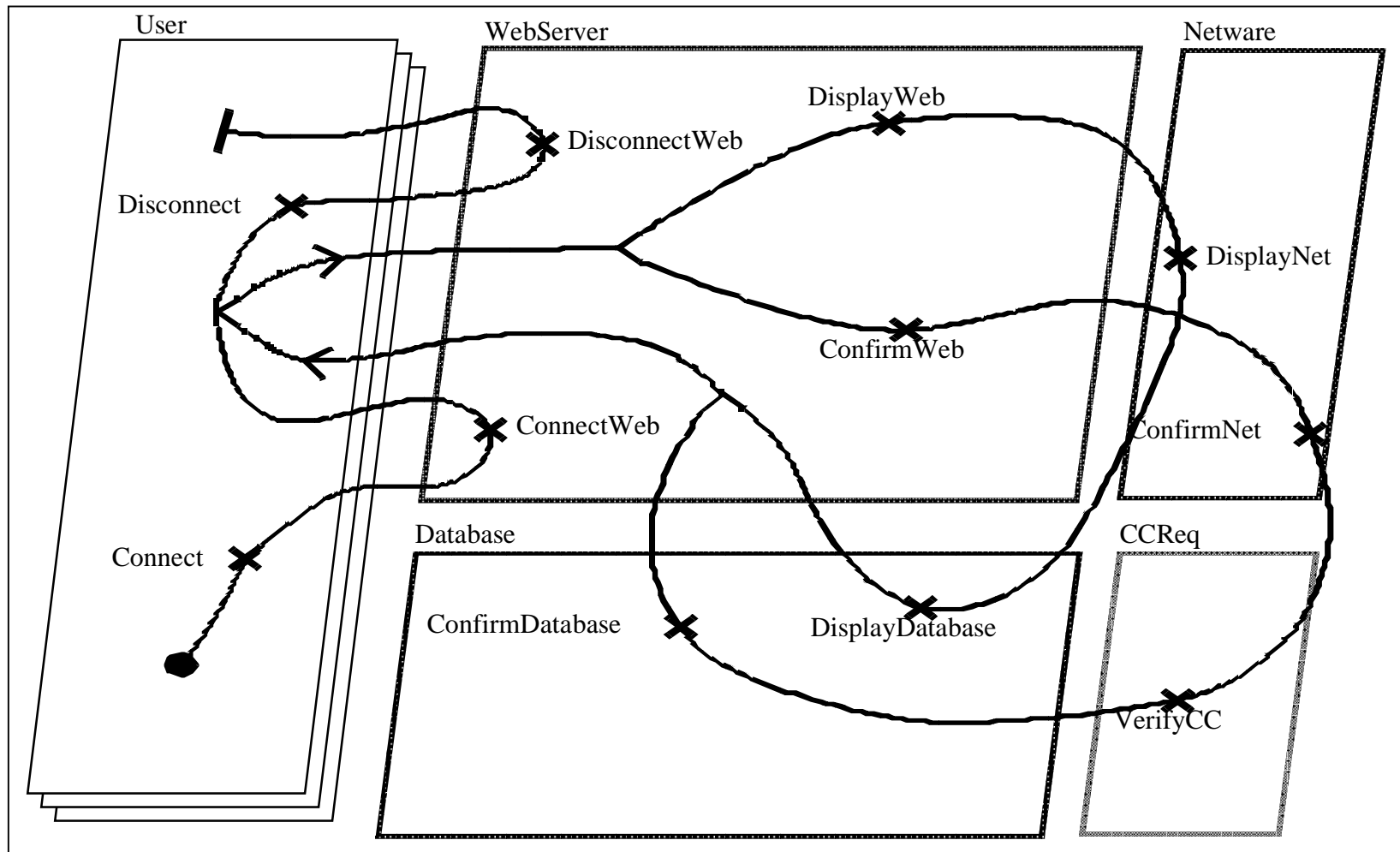


devices, tasks, entries, and activities

Creation of LQN Objects

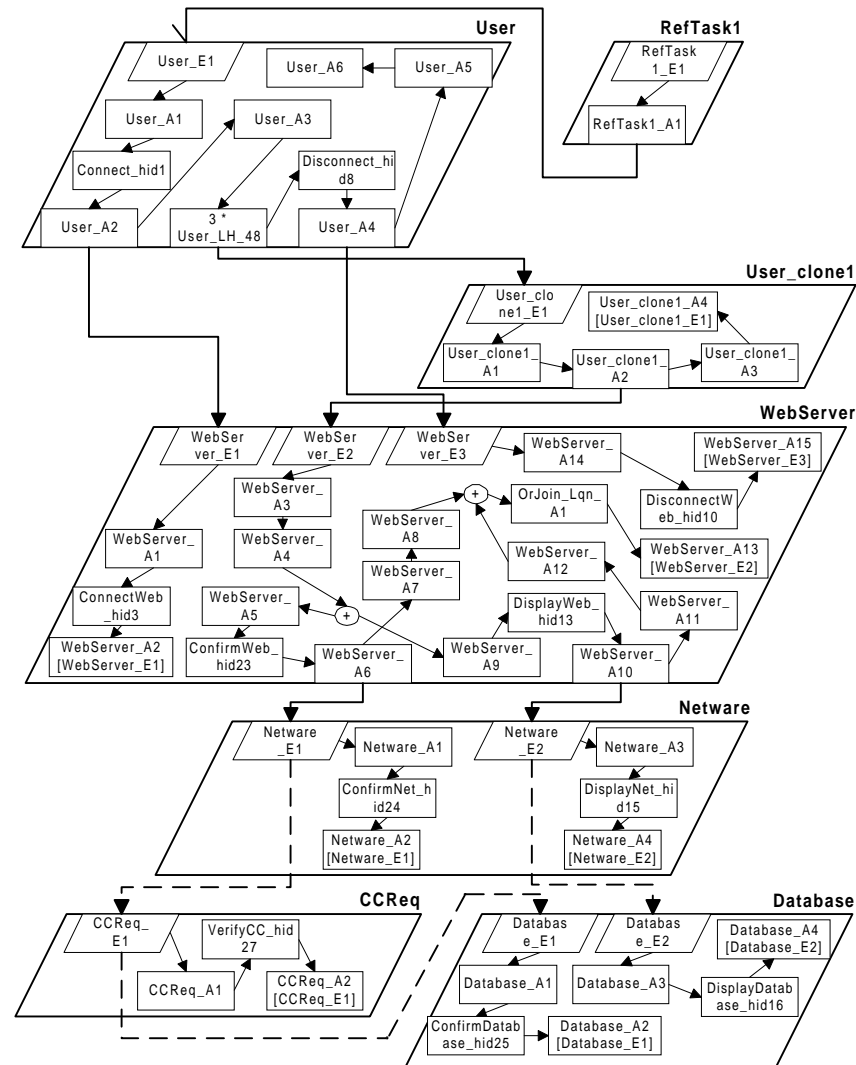
- LQN activities are assigned workload demands from UCM responsibilities
 - if demands are not specified in UCM, then default demands are assigned
- OR branches are assigned probabilities from UCM
 - if probabilities are not specified then they are assigned as equal
- arrival rates are assigned from the start points
 - if arrival rates are not specified then they are assigned default values

UCM for a Web-based Ticket Reservation System



UCM2LQN - Ticket Reservation System

Generated LQN Performance Model



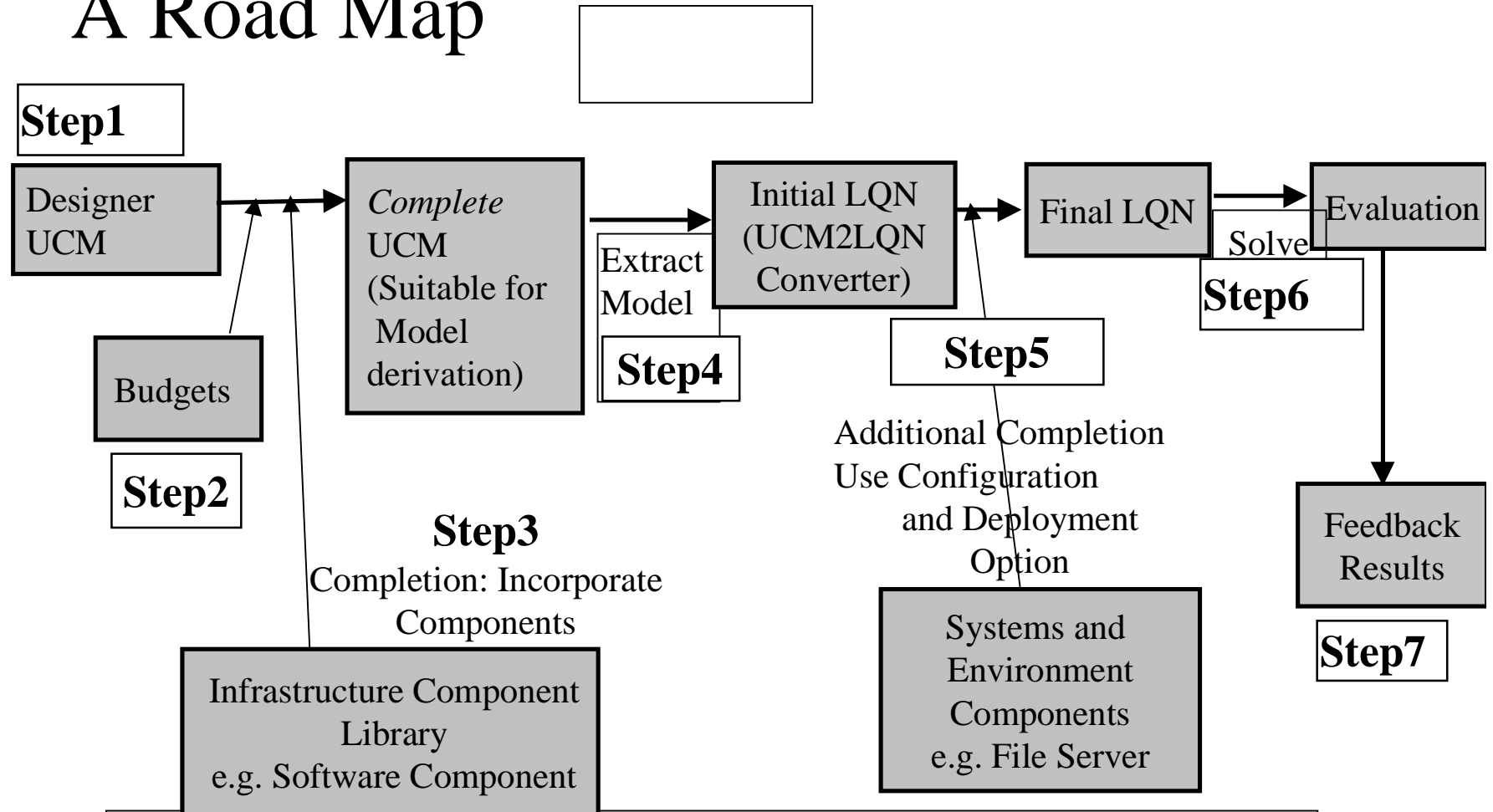
Techniques:

- (1) Budgeting for resource demands
- (2) Completions for missing information

Budgeting

- ***Problem:*** how to estimate resource demands for a function that has not yet been programmed
- ***Budget solution:*** treat it as budget for CPU time and other resources....
 - developer or group bargains with the architect for resources, commits to the budget
 - review, overruns, etc follow similar path as for financial budgets
 - a familiar solution in real-time... *extend it here for contention systems, by solving contention models*

Budget Analysis for contention systems: A Road Map



Budget Analysis Road Map

(2) Completions for missing information

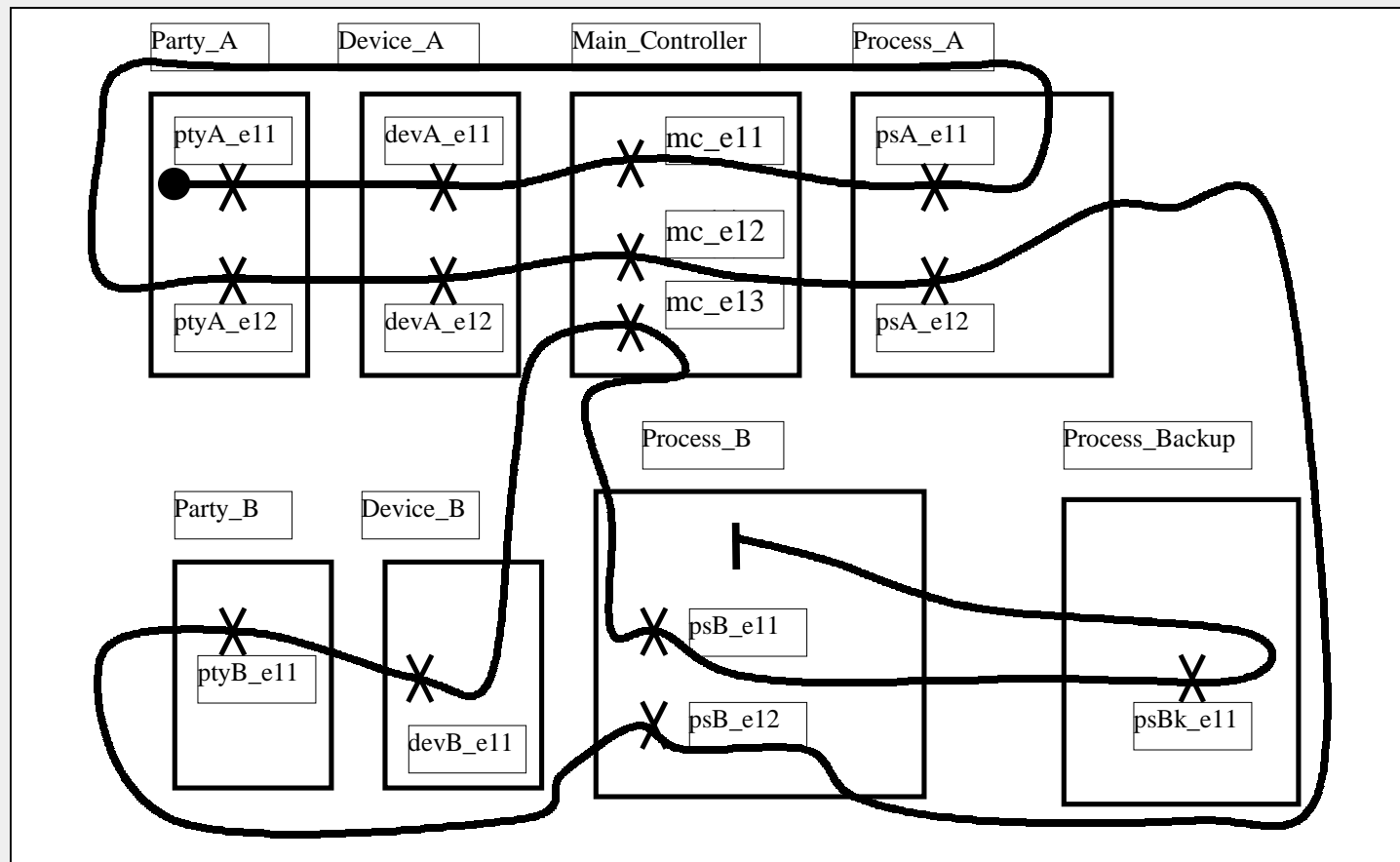
- Early specifications are incomplete
 - avoid unnecessary detail of infrastructure and surrounding functionality
 - inter-object and inter-process communications
 - communications protocols and networks
 - operating system functions
 - middleware
 - databases
 - web servers
 - file servers and networks
 - implied by group context, maybe by notes
- Automated and semi-automated completions
 - structure and performance attributes

Completions at the UCM level

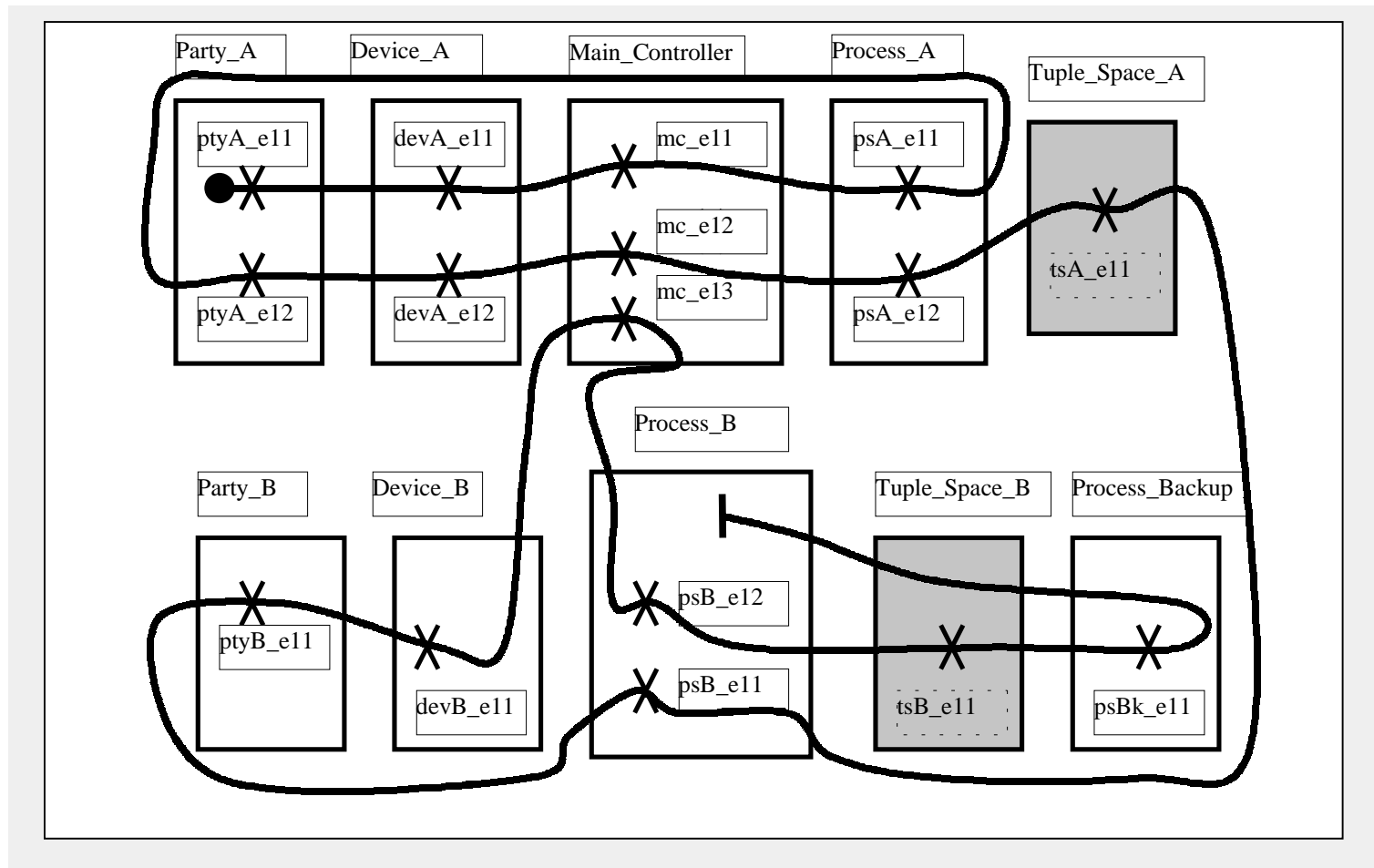
- using the stubbing mechanism of UCMs, add sub-scenarios for missing detail
 - import plugins (or even map fragments) from a library: pre-built standard behaviour patterns.
 - shared stubs for communications
 - questions of binding their component definitions into the components in the design
 - at present this is manual....
- potential automation: define environment choices for the system and generate choices for stubs or components

Example: Distributed Hand-off Protocol (Designer UCM)

- Process A hands a job to B which does a Backup

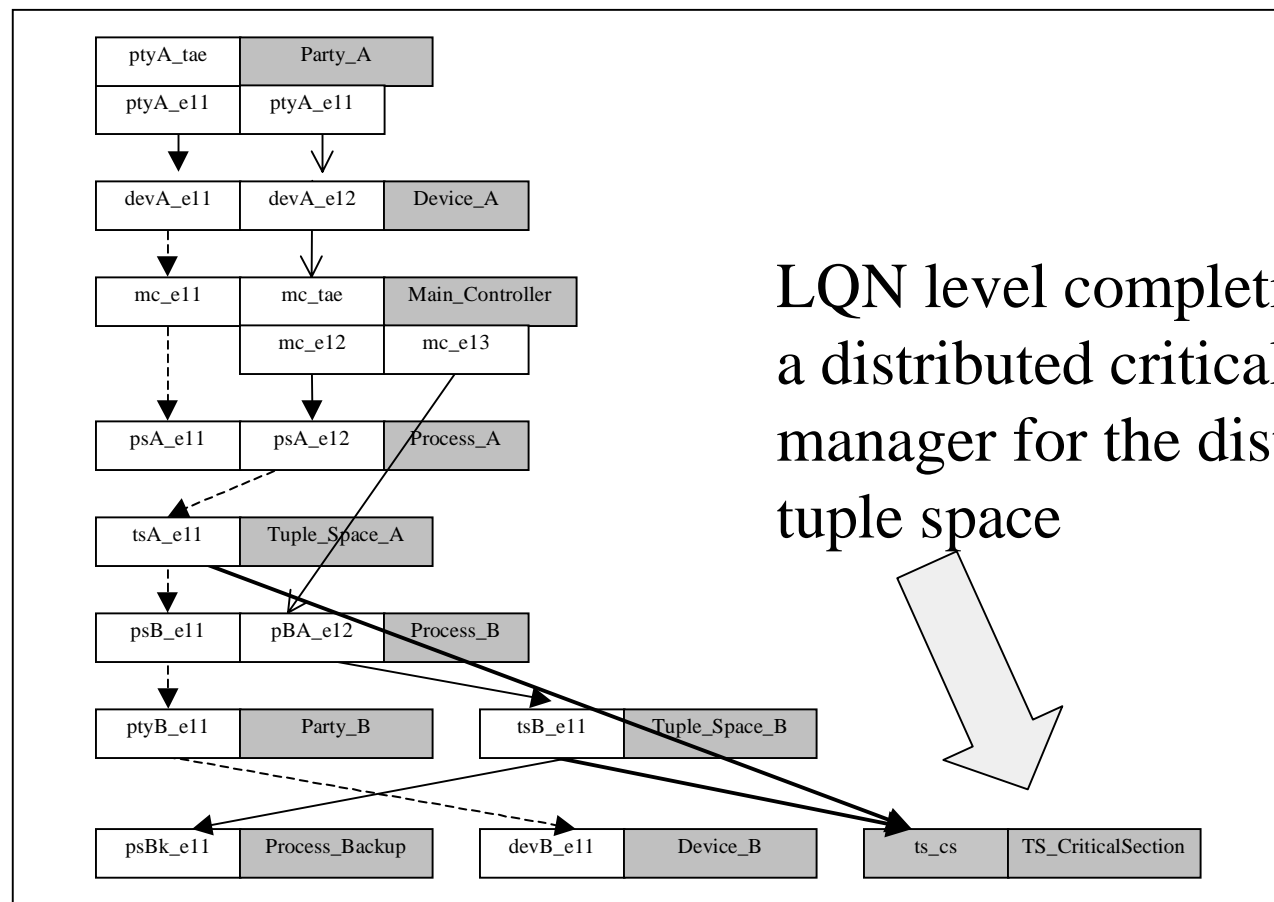


Example: Completion at the UCM level: communications between Process A and B by a *Distributed Tuple Space*



Example: Distributed Hand-off Protocol LQN Model

.... generated automatically; LQN completion (critical section)



LQN level completion is
 a distributed critical section
 manager for the distributed
 tuple space

Completions at the LQN model level

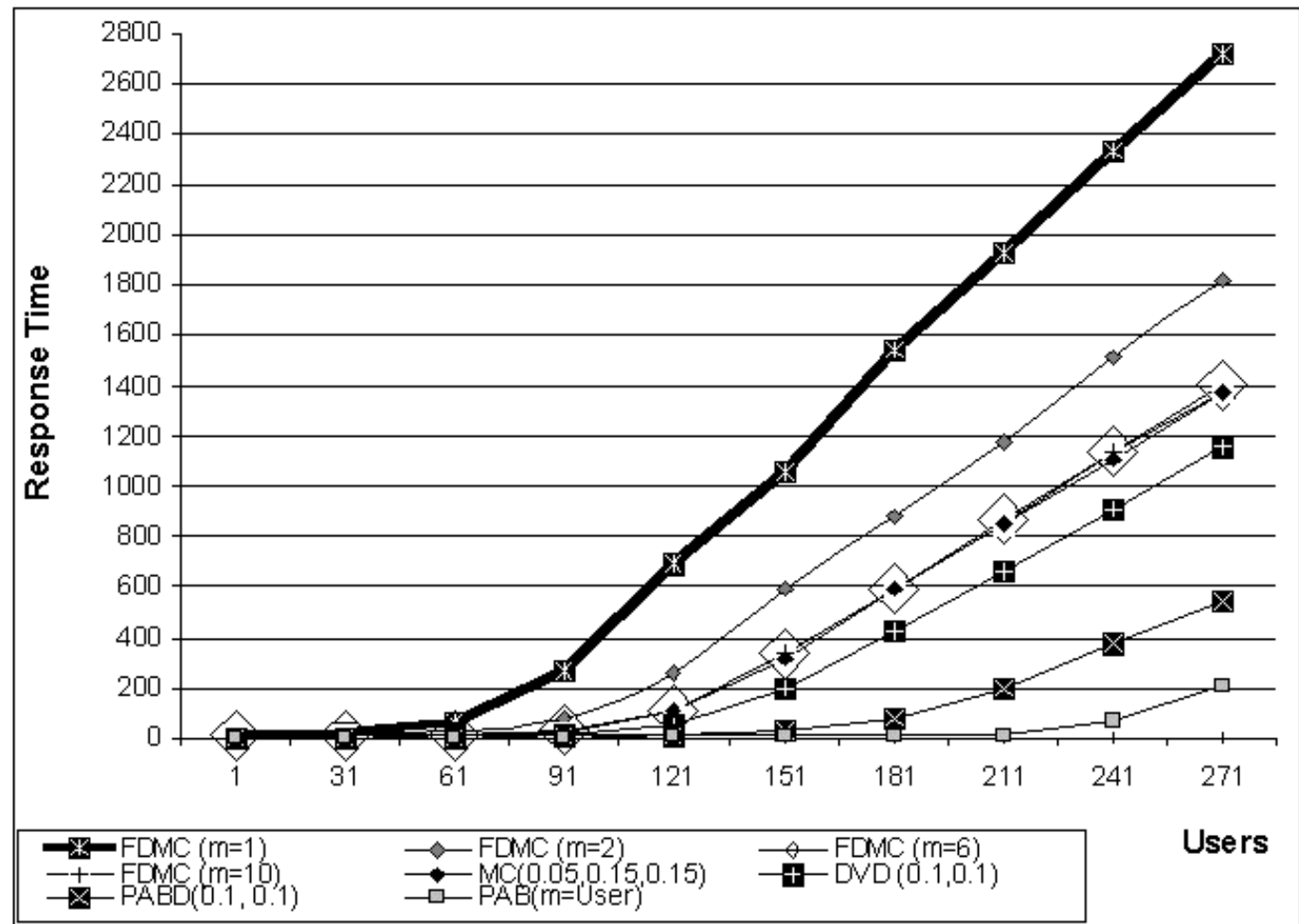
- System components can have pre-built submodels that can be patched together
 - LQN is architectural, so this is transparent
 - especially suitable for servers (web server, file server)
 - allows to experiment with alternatives

- Communications completions for representations of a network and its protocol layer execution
 - replace an interaction arc by a sub-system, or use of a sub-system.

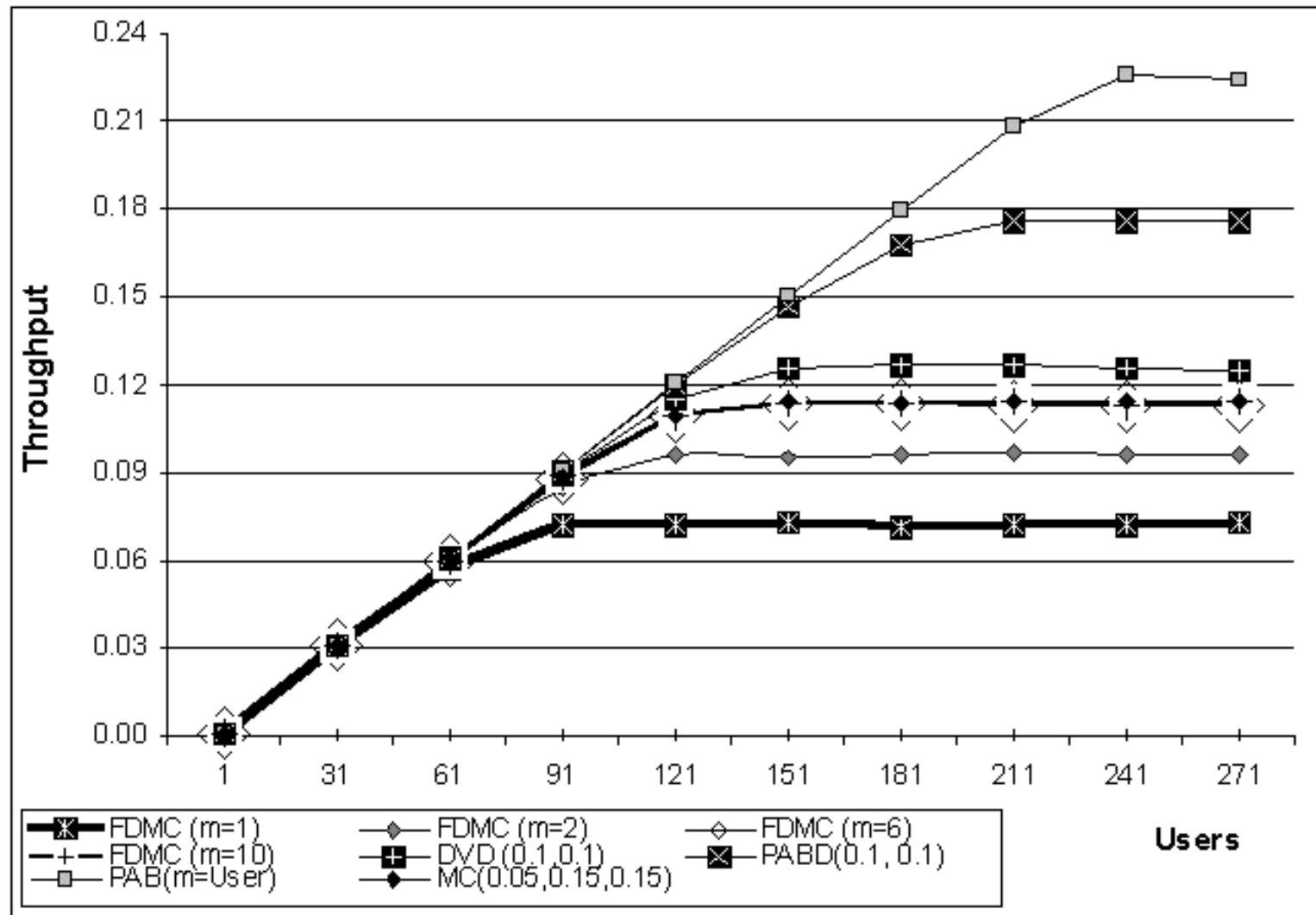
Handover example results: Mean delay increases with Users

....multi-threading reduces delay and sensitivity

.... faster execution has a smaller effect



Handover example results: Throughput increases with Users



Core E-PAD Methodology

- UCM specification of the system using UCMNav
- add workload parameters
 - based on workloads from similar systems
 - based on known parameters from existing components
 - possibly use a budgeting approach.....----->>>
- completions for missing detail
 - re-use from a library
- generate LQN model using UCM2LQN
- LQN-level completions
- solve LQN model using existing solvers (LQNS analytic solver, ParaSRVN simulator)

...Core E-PAD Methodology

- reason about performance impacts of the scenario and the architecture:
 - exploit concurrency and parallelism
 - identify potential performance bottlenecks
 - choose from alternative architectures, behaviours, components
 - evaluate scalability

