



Augmenting Architectural Modeling to Cope with Uncertainty

Orieta Celiku, David Garlan, and Bradley Schmerl
Carnegie Mellon University

5 November 2007

Motivation:

Relating Uncertainty to Architecture

- Coping with uncertainty at design time and a high level
 - understanding *expected* behavior even when we cannot know exact outcomes
- Working with architectural models
- Adopting existing specification and analysis approaches to specifying and analyzing architectural models

Today's Architecture Models

- Primarily focus on architectural properties that can be precisely defined
 - Assume complete information (at some level) about architectural behavior
- May use probabilistic uncertainty for very specific kinds of analysis
 - Limited to specific architectures and domains
 - E.g., queing-theoretic performance analysis

Our Approach

1. Augment property specification in ADLs with the ability to incorporate uncertainty in those properties
2. Make use of such properties for analysis such as architecture-based simulation and run-time detection of behavioral drift
3. Augment behavior descriptions to explicitly account for probabilistic behavior

Architecture Properties as Distributions

- Example:

Property type

NormalDistribution = **Record** [

mean : float;

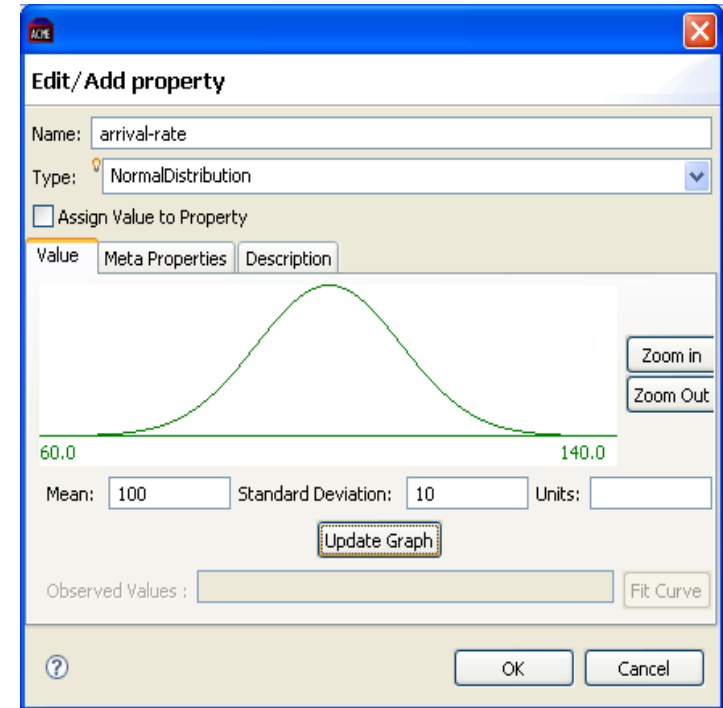
stddev : float;

]

...

property

arrival-rate : NormalDistribution =
[mean=100; stddev=10;];



- Such properties can be represented and visualized using standard mechanisms

Using Distributions in Analyses

- Properties specified as distributions can be used
 - as the basis of simulations
 - e.g., performance simulation and security simulation
 - to compare observed values with predicted distributions
 - e.g., to determine whether changes should be made to executing systems

Using Distributions in Analyses: (Monte-Carlo) Performance Simulation

The screenshot displays the 'View Report - 3 - Multithreading and Queuing Architecture Simulator' interface. It is divided into several sections:

- Evaluation Summary:** A table listing simulation parameters and results.

Property	Value
Scenario	Scenario1
Number of users	5
Transaction Generation Rate	3
Actual Simulation Load	
Actual Network Load	0
No. of System Transactions Generated	{ST1=24, ST2=24}
No. of System Transactions Completed	{ST1=24, ST2=24}
Average System Transaction Completion Time	156938
- System Diagram:** A flow diagram showing a 'Client' (yellow box) connected to a 'Server' (blue box), which is in turn connected to an 'Asset Database' (cylinder). Arrows indicate the direction of data flow.
- Properties Panel:** A configuration window with tabs for 'Performance Values' and 'Error Handling'.
 - Performance Values:**

Transaction Complexity	Very Simple	Simple	Average
Minimum Value	1.02	1.041	1.06
Maximum Value	1.03	1.05	1.07
 - System Resources:**
 - System Resources Consumed (in %): 5.0
 - Multithreaded
 - Queue
 - Max. Threads: 5
 - Queue Size: 100
- Error Handling Panel:**

Errors	Selected	Parameters	Value	Error Handling Mechanism
Process Crash	<input checked="" type="checkbox"/>	Successful system trans. (%)	99	Connect to another Thread, Log
Component Crash	<input type="checkbox"/>			

Augmenting Architectural Behavior with Probabilities

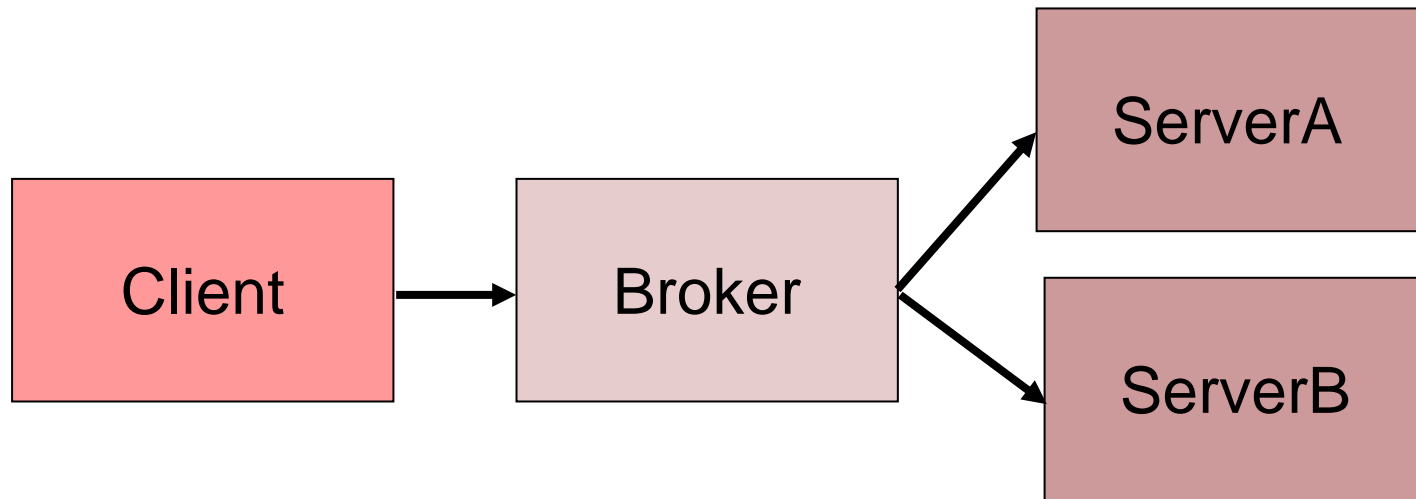
- Incorporate quantifiable uncertainty in behavioral descriptions of architectures
 - uncertainty expressible in terms of probabilistic distributions
- Enable incremental refinements of models:
 - making information more precise as we learn more about behaviors
 - extending specifications without invalidating previous analyses

Adopted Framework:

Probabilistic Action Systems (McIver 2006)

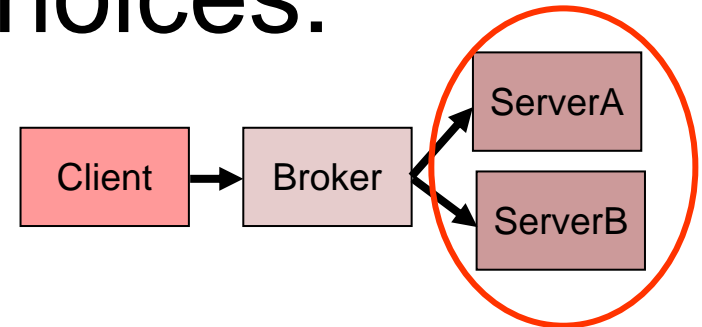
- Give access to explicit probabilistic choices (in addition to standard choices in behavior)
 - e.g., $\text{coin} := \text{heads} \frac{2}{3} \oplus \text{coin} := \text{tails}$
 - represent the frequency with which branches of a choice are executed
 - give rise to probabilistic assurances about properties (as opposed to absolute certainty)
- Semantics extends standard assertion-based reasoning with probabilistic analysis over a suitable probabilistic space on computation paths
 - can reason about probability of a property holding, or more generally
 - expected value of some random variable of interest

Modeling with Probabilistic Action Systems: A Client-Server System



- Action Systems Client, Broker, ServerA, and ServerB execute in parallel
- Each action system is a collection of guarded actions executing in parallel

Explicit Probabilistic Choices: Servers



ServerA ==

var *sa*: {*wait*,*servicing*,*served*}

initially *sa* := *wait*

serviceArequest: (*sa* = *wait*) → *sa* := *servicing*

serviceA: (*sa* = *servicing*) → *sa* := *served* $p_A \oplus$ *skip*

serviceAreceive: (*sa* = *served*) → *sa* := *wait*

- Server A chooses to delay serving with probability $1-p_A$
- ServerB is specified analogously and chooses to delay serving with probability $1-p_B$

Probabilistic Properties

- Properties we can reason about using AS semantics:
 - The probability of establishing a postcondition
 - e.g. “For p_A and p_B bounded away from 0, the Client will eventually be served with probability 1”
 - Probabilistic versions of temporal properties,
 - guarantees expressed as “the least probability with which the property holds”

Nondeterministic Choices: Broker

Broker ==

var *r*. {*listen,serve,serving,served*}

initially *r* := *listen*

request: $(r = \textit{listen}) \rightarrow r := \textit{serve}$

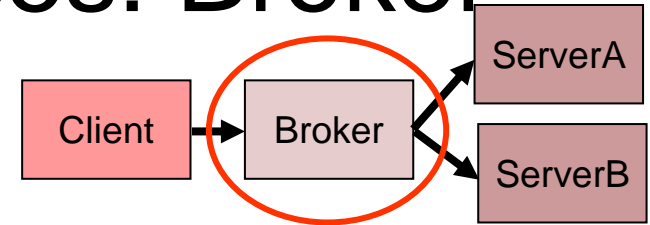
serviceArequest: $(r = \textit{serve}) \rightarrow r := \textit{serving}$

serviceBrequest: $(r = \textit{serve}) \rightarrow r := \textit{serving}$

serviceAreceive: $(r = \textit{serving}) \rightarrow r := \textit{served}$

serviceBreceive: $(r = \textit{serving}) \rightarrow r := \textit{served}$

receive: $(r = \textit{served}) \rightarrow r := \textit{listen}$



Two or more enabled actions give rise to (standard) nondeterministic behavior

Nondeterministic Choices: Room for Refinement

Broker ==

...

serviceArequest. ($r = \text{serve}$) $\rightarrow r := \text{serving}$

serviceBrequest. ($r = \text{serve}$) $\rightarrow r := \text{serving}$

...

When both servers are ready, we have no guarantees about which one will be chosen

- the best we can do is assume that “the worse” server is chosen
- we can only guarantee service within the first “tick” with probability **$\min(p_A, p_B)$**



Nondeterministic Choices: Refinement

- We could do better if we flipped a coin instead

Broker ==

...

whichserver. ($r = \text{serve}$) $\rightarrow \text{which} = A \oplus_{1/2} \text{which} = B$

serviceArequest. ($\text{which} = A$) $\rightarrow r := \text{serving}$

serviceBrequest. ($\text{which} = B$) $\rightarrow r := \text{serving}$

...

- A valid implementation of arbitrary choice is one in which a coin is used to decide which branch to execute
 - probabilistic choices “average” results
- We could postpone the decision until we know how p_A and p_B are related
 - use a biased coin favoring the faster server

On Going Research

- Tailoring the described formalism to better express architectural notions and concerns
 - e.g. connectors, hierarchy, ports/roles
- Case studies
- Tool support for analyses