Software Architectures

Presenter: Marwan Abi-Antoun

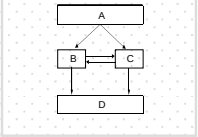
Slides Courtesy of Professor David Garlan

Why Document Architecture?

- Blueprint for the system
  - Artifact for early analysis
  - Primary carrier of quality attributes
  - Key to post-deployment maintenance and enhancement
- Documentation speaks for the architect, today and 20 years from today
  - As long as the system is built, maintained, and evolved according to its documented architecture

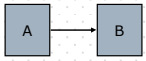
What is Wrong Today?

- In practice today's documentation consists of
  - Ambiguous box-and-line diagrams
  - Inconsistent use of notations
  - Confusing combinations of viewtypes
- Many things are left unspecified:
  - What kind of elements?
  - What kind of relations?
  - What do the boxes and arrows mean?
  - What is the significance of the layout?



What could the arrow mean?

- Many possibilities
  - A passes control to B
  - A passes data to B
  - A gets a value from B
  - A streams data to B
  - A sends a message to B
  - A creates B
  - …



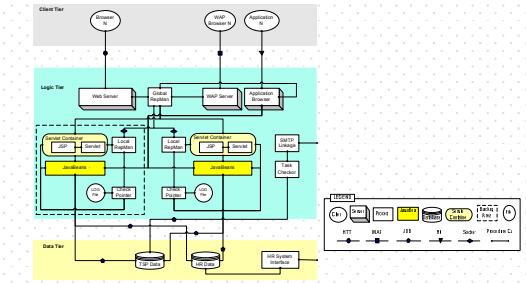
Guidelines: Avoiding Ambiguity

- **Always include a legend**
- Define precisely what the boxes mean
- Define precisely what the lines mean
- Don't mix viewtypes unintentionally
  - Recall: Module (classes), C&C (components)
- Supplement graphics with explanation
  - Very important: rationale (architectural intent)
- Do not try to do too much in one diagram
  - Each view of architecture should fit on a page
  - Use hierarchy

Technique: Multiple Views

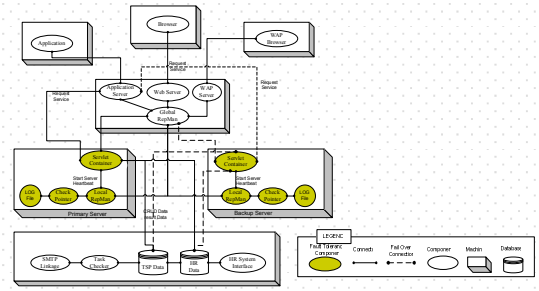
- Separate views to manage complexity
  - Provide clean separation of concerns
  - But lead to the problem of relating them and finding inconsistencies between them!
- Showing how views relate to each other can help better understand the architecture

Overlay of C&C with Tiers



7

Overlay of Allocation and C&C



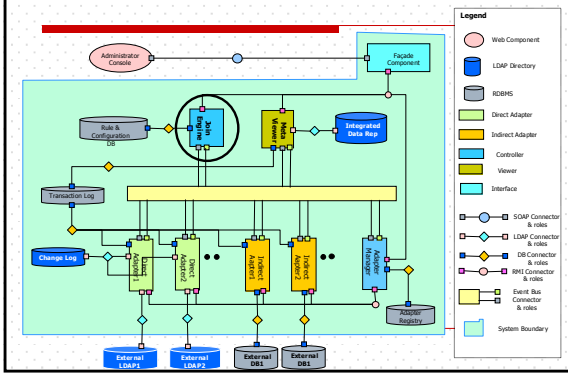
8

Technique: Hierarchy

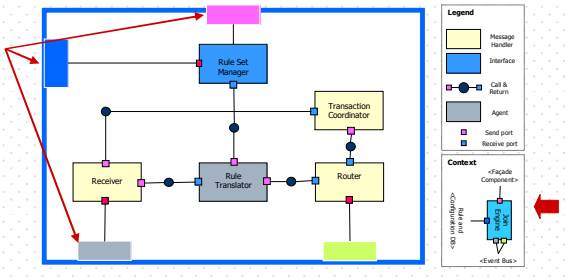
- Use hierarchy to define elements in more detail in separate views
- Helps keep an architectural description manageable

9

Top-level C&C View



Showing Details of Component



11

Analyzing Architectures

- To more precisely document an architecture, you can use an Architecture Description Language (ADL)
- We're going to illustrate one such ADL, Acme:
  - Supports hierarchical design
  - Support architectural styles
  - Supports expressing/analyzing extra-functional properties (performance, reliability, etc.)
  - Does not support checking of conformance of implementation to architecture

12

Many Architecture Description Languages

- Each comes with different analysis capabilities
  - Rapide: events with simulation and animation
  - UniCon: emphasizing heterogeneity and compilation
  - Wright: formal specification of connectors
  - Aesop/Acme: style-specific arch design languages
  - Darwin: service-oriented architectures
  - SADL: architectural refinement
  - Meta-H: arch description for avionics domain
  - C-2: arch style using implicit invocation

AcmeStudio Demo

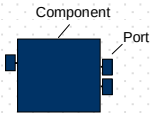
- Create/Modify a Family
- Create/Modify a System

Acme Primer

- Family
  - Component Types, Port Types, Connector Types, …
- Components
  - Ports
- Connectors
  - Roles
- Attachments
- System
- Representations
  - Bindings

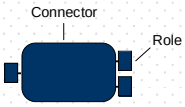
Components and Ports

- Components
  - Represent the computational elements and data stores of a system.
- Ports
  - Are the points of interaction between a component and its environment.



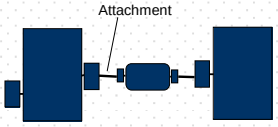
Connectors and Roles

- Connectors
  - Represent interactions between components such as method calls or an SQL connection
- The interface of a connector is defined as a set of roles (think of it as the "protocol")



Attachments

- Attachments
  - Links a component port to a connector role.

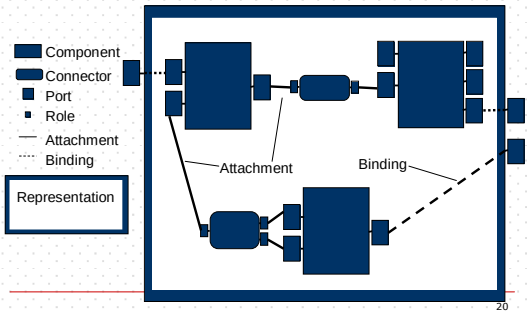


System

- System
  - A set of components, a set of connectors, and a set of attachments, **assigned a set of styles and types**

19

Representations and Bindings



20

Kinds of Architectural Analyses

- Consistency
  - Do the parts fit together?
- Completeness
  - Are parts missing?
- Refinement
  - Can one architecture be substituted for another?
- Verification
  - Does an implementation conform to the architecture?
- System-wide behavior, performance, reliability, etc.
  - What is the aggregate behavior, performance, reliability of a system, given the .. of the parts?
- Impact analysis
  - Architectural Tradeoff Analysis Method (ATAM)
- Others?

21

Analysis Strategies

- Use existing specification languages & calculi
  - Examples: CSP, Queuing Theory, etc
  - Advantages: well understood, tools, reuse
  - Disadvantages: may not be expressive; may require a lot of initial "context building" before you can do anything useful
- Develop new architectural specification languages & reasoning techniques
  - Examples: use Wright for checking behavior
  - Advantages: good match to the problem
  - Disadvantages: learning curve, proliferation of languages and tools

22

Analysis: consistency & completeness

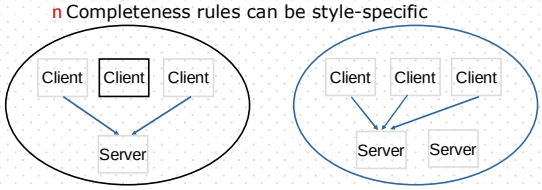
- Consistency: do the parts fit together?
  - analog of type checking
  - depends on what you say about the parts
  - behavior example: does the behavior of a component conform to the protocols of a connector to which it is attached?

- Completeness: are any parts missing?
  - connector roles?
  - unattached ports?
  - missing functionality?

23

Example of a completeness check

- Example completeness rule: All clients need to be attached to at least one server.
  - Client with no server is incomplete
  - … but server with no client is fine.
  - Completeness rules can be style-specific



Incomplete! Complete

24

Example Consistency Checking

- Key idea: Acme can capture complex constraints about an architectural style
- AcmeStudio (the tool) can be used to automatically check an architecture's conformance to these constraints.
- Example: Topological constraints of a complex family of NASA systems.

25

Specifying Architectural Constraints

- Acme includes a constraint language
  - Based on first-order predicate logic
  - Augmented with architecture-specific predicates
    - connected(comp1, comp2), self.ports, self.components, …
- Examples:
  - A particular type of component can only have particular types of ports
  - Property values must have certain values
- Two types of constraints
  - Invariants must never be violated
  - Heuristics may be selectively violated

26

Recall Acme Constraints

```
Component Type naïve-client = {

    Port Request = {
          Property protocol = rpc-client };

    Property request-rate : integer
       << default = 0; units = "rate-per-sec" >>;

    Invariant forall p in self.Ports |
                 (p.protocol = rpc-client);
    Invariant size(Ports) <= 5;
    Invariant request-rate >= 0;
    Heuristic request-rate <= 100;
}
```

27

Consistency Example: MDS Architectural Style

- MDS defines an architectural framework for a family of NASA systems
  - System of architectural component types
  - Rules on how they can be connected
- Checking/ensuring conformance to MDS is an important and hard problem
  - Many rules, many components, complex interconnection topology

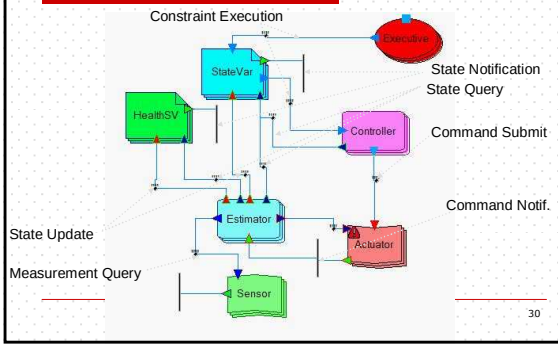
28

Acme MDS Architectural Style

- To specify the MDS style requires
  - 8 Component types (sensor, actuator, estimator …)
  - 12 Connector types (measurement query, command submit, state update)
  - 22 Interface types
- MDS rules defined using first order predicate logic
  - Ten rules in English from MDS designers become 38 checkable predicates

29

Acme System in the MDS Style



30

Example MDS Rule

- As specified by MDS designers:

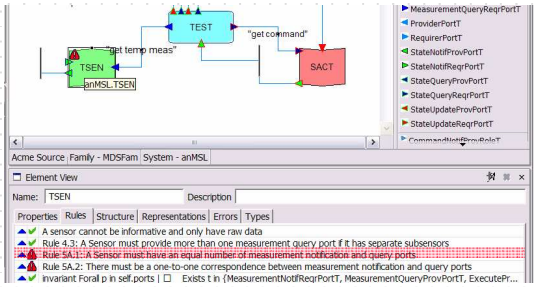
  For any given Sensor, the number of Measurement Notification ports must be equal to the number of Measurement Query ports (rule R5A).

- Acme rule (associated with the sensor component type):

```
numberOfPorts (self, MeasurementNotifReqrPortT) ==
  numberOfPorts (self, MeasurementQueryProvPortT)
```

31

Checking the MDS Rule



32

More MDS Rules

- Rule 4:"Every estimator requires 0 or more Measurement Query ports. It can be 0 if estimator does not need/use measurements to make estimates, as in the case of estimation based solely on commands submitted and/or other states. Every sensor provides one or more Measurement Query ports. *It can be more than one if the sensor has separate sub-sensors and there is a desire to manage the measurement histories separately.* For each sensor provided port there can be zero or more estimators connected to it. It can be zero if the measurement is simply raw data to be transported such as a science image. It can be more than one if the measurements are informative in the estimation of more than one state variable."

33

Checking More MDS Rules

- As specified by MDS designers:

  "*…It can be more than one if the sensor has separate sub-sensors and there is a desire to manage the measurement histories separately*…."

- Acme rule (associated with the sensor component type):

  (numberOfPorts(self, MeasurementQueryPort) > 1) à
  self.manageHistoriesSeparately AND
  hasCommandableSubunits(self));

  where

  hasCommandableSubunits = …

34

Analyzing system-wide properties

- Key idea: calculate properties of a system, given properties of its parts
- Different kinds of properties will have different calculi for compositionality
- Usually depends on using a specific style
  - *Example 1:* queuing theory can be used to calculate overall throughputs and latencies if use asynchronous message passing style
  - *Example 2:* reliability block diagrams can be used to determine aggregate reliability from the parts, for certain styles

35

Performance Analysis Alternatives

- Measurement
  - Most accurate; modifications difficult
  - Requires existing system, workload
- Simulation
  - Accuracy and ease of modifications vary
  - Requires existing simulator, workload
- Analytical model
  - Back-of-the-envelope accuracy in design phase
  - Nontrivial systems require nontrivial calculation; analysis tools can help.

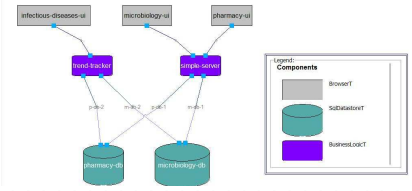
36

Architectural Performance Analysis

- Try out various "what-if" scenarios,
- Obvious ways to deal with a bottleneck component:
  - replicate it,
  - speed it up, or
  - reduce the demand on the system.
- Options may vary in expense and difficulty, or even feasibility

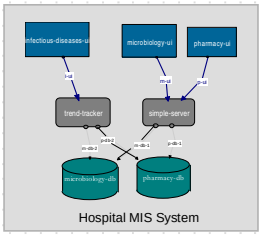
Example 1: Performance predictions

Problem: Evaluate performance of a three-tier medical informatics system



Evaluate performance of a three-tier medical informatics system

- Can the servers handle the expected demand?
- Will the average response time meet requirements?
- How large should the buffers be?
- Highest demand the servers can handle?
- Which component is the bottleneck?
- How would server/database replication affect this?



Hospital MIS System

Queuing Network Theory 101

- Basic units of a queuing network:
  - Queues: A buffer with some queuing discipline (e.g., FIFO, round robin)
  - Service centers: provides some necessary service
- Service center:
  - Has a queue containing jobs to process
  - Can be replicated: i.e., *m* identical providers of service, which draw their jobs from a single queue
- A queuing network is an interconnected group of these queues.
  - Jobs enter the network, receive service at service centers, and leave

Queuing Network Theory 102

- Given for each service center
  - average arrival rate of jobs
  - average service time to process a job
- Assumptions
  - asynchronous arrival of jobs in queues
  - exponential rates, independence
  - each job exists in exactly one place at a time

Queuing Network Predictions

- Utilization (fraction of time the service center is occupied)
- Average time a job spends waiting in the queue
- Average queue length
- Probability that the queue length is n
- Latency: the time for a job to be completely processed
- Throughput: the rate at which jobs are processed
- Number of outstanding jobs in the system
- Most-utilized service centers, i.e., possible bottlenecks
- Whether the system is stable or overloaded
  - In an overloaded system, queue grows faster than jobs can be processed; the server cannot keep up
  - For a queue implemented as a buffer of length B, the rate at which incoming jobs are discarded due to buffer overflow (drop rate)

Naive Application to Architectures

- Define an architectural style in which
  - Service centers → distributed processing components
  - Transmission lines → directional asynchronous message passing connectors
- Associate service times with components and arrival rates with system
- Use Queueing Networks to calculate derived values for the components and the system

Complicating Factors

- Cycles
  - jobs may pass through same component several times before exiting
- Autonomous clients
  - generate jobs by themselves
- Delays in connectors
  - present for architectures of real systems
- Replication
  - meaning and effect on calculations

Modeling Performance in Acme

```
Family PerformanceFam = {
  Component Type PerformanceComponent = {
    perf-replication : int;
    perf-overloaded : boolean;
    perf-serviceTime : float;
    perf-responseTime : float;
    perf-utilization : float;
    …
  }
  Connector Type PerformanceConnector = {
    perf-delayTime : float;
    …
  }
  …
}
```


Modeling Three-Tier Family in Acme

```
Family WebThreeTier extends PerformanceFam with {
  Component Type BrowserT extends
      PerformanceComponent with {…}
  Component Type SqlDataStore extends
      PerformanceComponent with {…}
  Component Type BusinessLogic extends
      PerformanceComponent with {…}

  …
}
```


Modeling the System

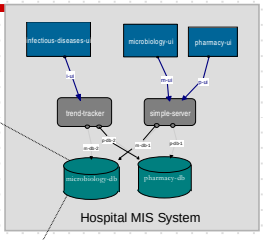
```
System med-informatics : ThreeTierFam = {

  Component infectious-diseases-ui : BrowserT =
    {…};
  Component microbiology-ui : BrowserT = {…};
  Component pharmacy-ui : BrowserT = {…};

  Component simple-server : ServerT = {
    Property perf-replication = 10;
    … };

  Component trend-tracker : ServerT = {
    Property perf-replication = 1;
    … };

  . . .
```


QN Example: Local Properties

User defines local performance properties by:

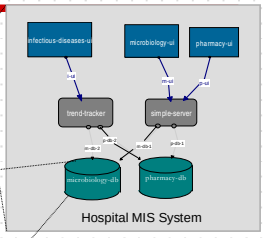
- Simulation
- Guesstimate
- Measured values

```
Component microbiology-db = {
  Ports {…};
  Property localPerformance = [
    replication = 2;
    serviceTime = 1000ms;
    outputMessagePaths = {…};
  ];
  Property computedPerformance = [
    … ?
  ];
};
```



QN Example: Emergent Global Properties

System hospitalMIS = { ...
Property systemPerformance = [
avgResponseTime = 2784ms;
avgSystemMsgs = 2.342;];
Component microbiology-db = {
Ports {...};
Property localPerformance = [...];
Property computedPerformance = [
avgResponseTime = 2143ms;
avgQueueLength = 0.923;
avgUtilization = 0.48;
overloaded = false;
]; };
};



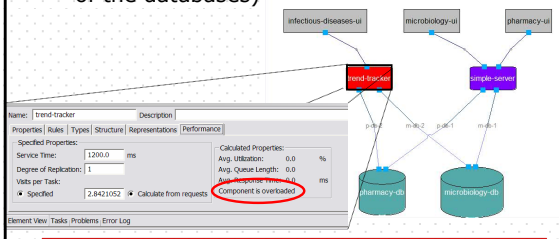
Tool computes system-wide performance properties based on configuration

Architectural Analysis in Acme

- In Acme (the language):
  - Capture performance-related properties in a family
  - Systems desiring performance analysis (and satisfy the assumptions) use this family
  - Acme properties
  - Acme constraints
- In AcmeStudio (the tool):
  - Calculate the values related to the properties

Results of Performance Analysis

- The "trend-tracker" is overloaded (but neither of the databases)



What can we do about overloading?

Scaling Strategies

- Goal of scaling: eliminate performance bottlenecks in application
- **Vertical partitioning**
  - Add cache to process application requests more quickly
  - Ex: cache some data (e.g., list of countries) instead of hitting database
- **Vertical scaling**
  - *Buy a bigger machine*
  - More expensive from a hardware standpoint but cost less to maintain
- **Horizontal partitioning**
  - Partition application onto different servers.
  - For example: Split off reporting server onto another server
  - Requires careful thought about slicing the application
- **Horizontal scaling**
  - *Buy a farm of identical machines*
  - Drawback: you cannot store state information on any given machine
  - Cheaper from a hardware standpoint but cost more to maintain

Example 2: Reliability predictions

- Basic idea: apply Reliability Block Diagrams to Software Architecture
  - Reuses well developed reliability model
  - Slightly modified to work with Software Architectures
- Initial information includes:
  - reliability of individual components
  - topology of interaction, concurrency, replication
- From this we calculate expected reliability of the system as a whole.

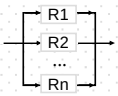
Reliability model for SW Architecture

- Reliability defined as $R = e^{-\lambda T}$ where:
  - λ is a component's failure rate
  - T is the time period over which reliability is measured

Reliability Block Diagrams Example

$$R_{sys} = \prod_{i=1}^{n} R_i$$

Serial Composition



$$R_{sys} = 1 - \prod_{i=1}^{n} (1 - R_i)$$

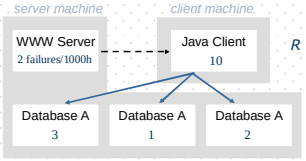
Parallel Composition

Reliability Block Diagram (RBD) Complications

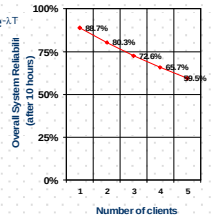
- Simple model has some complications
  - Concurrency
  - Distribution
  - Dynamism
  - Connectors may be unreliable
- These complications are addressed for many (but not necessarily all) styles.
  - For details, see [AbdAllah97] Abd-Allah, Ahmed, "Extending Reliability Block Diagrams to Software Architectures", USC Technical Report USC-CSE-97-501

Detailed RBD Example

Modeling a Java/WWW-based client-server system*



$R = e^{-\lambda T}$



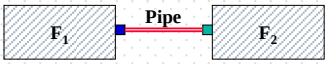
$$\lambda_{server} = \frac{2+3+1+2}{4} = 2$$

$$\lambda_{client} = 10$$

$$\lambda_{system} = \lambda_{server} + n\lambda_{client} \quad (for\ n \geq 0)$$

* Example borrowed from [AbdAllah97]

Specifying Architectural Behavior



- Which is the reading/writing end of the pipe?
- Is writing synchronous?
- What if F_2 tries to read and the pipe is empty?
- Can F_1 choose to stop writing?
- Can F_2 choose to stop reading without consuming all of the data?
- If F_1 closes the pipe, can it start writing again?
- If F_2 never reads, can F1 write indefinitely?

Questions