ANALYZING SOFTWARE ARCHITECTURES:
A SEMANTIC MODEL

Ali Mili, NJIT
NII, Tokyo, Japan
January 10, 2012
HAPPY NEW YEAR
PLAN

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
- An Attribute Grammar for ACME+
- Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
SOFTWARE QUALITY ATTRIBUTES

A talk on software architecture starts with a discussion of software quality attributes... Why?

- Architectural decisions determine quality (non functional) attributes.
  - Whereas design/programming decisions determine functional attributes/properties.

- Architectural decisions are taken according to selected attributes.
  - Prioritizing attributes as a major system level decision. We cannot have them all.

- Study of software architectures inseparable from the study of quality (non functional) attributes.
# Software Quality Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Measure</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Hrs/days</td>
<td>Failure Freedom</td>
</tr>
<tr>
<td>Safety</td>
<td>Hrs/days</td>
<td>Freedom from catastrophic failure</td>
</tr>
<tr>
<td>Security</td>
<td>Hrs/days</td>
<td>Freedom from security breaches</td>
</tr>
<tr>
<td>Availability</td>
<td>Percent.</td>
<td>Ratio of operational time</td>
</tr>
<tr>
<td>Throughput</td>
<td>Trans/sec</td>
<td>Volume of processing</td>
</tr>
<tr>
<td>Maintainability</td>
<td>PM/loc yr</td>
<td>Unitary Cost of Maintenance / year</td>
</tr>
<tr>
<td>Response Time</td>
<td>Sec</td>
<td>Time between query and response</td>
</tr>
<tr>
<td>Portability</td>
<td>PM/ migr</td>
<td>Ease of migration between platforms</td>
</tr>
<tr>
<td>Failure Prob</td>
<td>Prob</td>
<td>Probability of failure/ unit of op time</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>Qualitat.</td>
<td>Ease of Use, even an adult can use it</td>
</tr>
<tr>
<td>Ease of Learning</td>
<td>Hrs</td>
<td>Required Effort to learn</td>
</tr>
<tr>
<td>Modularity</td>
<td>Cpl+Coh</td>
<td>Coupling and Cohesion</td>
</tr>
</tbody>
</table>
# Software Quality Attributes

From application domain to quality attributes

<table>
<thead>
<tr>
<th>Application</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart Monitor</td>
<td></td>
</tr>
<tr>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Online Banking</td>
<td></td>
</tr>
<tr>
<td>Batch Banking</td>
<td></td>
</tr>
<tr>
<td>Social Media</td>
<td></td>
</tr>
<tr>
<td>Incomplete Specs</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
</tr>
</tbody>
</table>

Menu

- Reliability
- Safety
- Security
- Availability
- Throughput
- Maintainability
- Response Time
- Portability
- Failure Prob
- Ease of Use
- Ease of Learning
- Modularity
# Software Quality Attributes

## From application domain to quality attributes

<table>
<thead>
<tr>
<th>Application</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart Monitor</td>
<td>Safety</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Availability</td>
</tr>
<tr>
<td>Online Banking</td>
<td>Reliability</td>
</tr>
<tr>
<td>Batch Banking</td>
<td>Throughput</td>
</tr>
<tr>
<td>Social Media</td>
<td>Security</td>
</tr>
<tr>
<td>Incomplete Specs</td>
<td>Maintainability</td>
</tr>
<tr>
<td>Compiler</td>
<td>Portability</td>
</tr>
</tbody>
</table>
# Software Quality Attributes

From application domain to quality attributes

<table>
<thead>
<tr>
<th>Application</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneous App</td>
<td></td>
</tr>
<tr>
<td>Flight Info Server</td>
<td></td>
</tr>
<tr>
<td>US Census 2020</td>
<td></td>
</tr>
<tr>
<td>Office Automation</td>
<td></td>
</tr>
<tr>
<td>Flight Control</td>
<td></td>
</tr>
<tr>
<td>Flight Simulator</td>
<td></td>
</tr>
<tr>
<td>Database Query</td>
<td></td>
</tr>
</tbody>
</table>
# SOFTWARE QUALITY ATTRIBUTES

From application domain to quality attributes

<table>
<thead>
<tr>
<th>Application</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneous App</td>
<td>Modularity</td>
</tr>
<tr>
<td>Flight Info Server</td>
<td>Ease of use</td>
</tr>
<tr>
<td>US Census 2020</td>
<td>Throughput</td>
</tr>
<tr>
<td>Office Automation</td>
<td>Ease of Learning</td>
</tr>
<tr>
<td>Flight Control</td>
<td>Safety</td>
</tr>
<tr>
<td>Flight Simulation</td>
<td>Reliability</td>
</tr>
<tr>
<td>Database Query</td>
<td>Response Time</td>
</tr>
</tbody>
</table>

Menu

- Reliability
- Safety
- Security
- Availability
- Throughput
- Maintainability
- Response Time
- Portability
- Failure Prob
- Ease of Use
- Ease of Learning
- Modularity
PLAN

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
- An Attribute Grammar for ACME+
- Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
Software architecture consists of the rules and principles for how a system is decomposed into its component parts, the rationale for how responsibilities are allocated among those parts, and the policies and mechanisms that coordinate the interactions between those parts as they collaborate to fulfill the purpose of the system. Software architecture is at once the partitioning of a system into its significant elements, and the organization and integration of those elements into a cohesive whole.
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLes

Architectural Models: Used to document an architectural design.

- **Static structural model** that shows the major system components. But there is more to architecture than structure (Boxes and Arrows).
- **Dynamic process model** that shows the process structure of the system. How control flows in the system.
- **Interface model** that defines sub-system interfaces.
- **Relationships model** such as a data-flow model that shows sub-system relationships.
- **Distribution model** that shows how sub-systems are distributed across computers.
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

Three important characteristics of an Architecture:
- Structure
- Communication
- Control
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

System Structure
- **Module Structure**: Modules and Dependencies.
- **Conceptual Structure**: Functional units and data flow.
- **Process Structure**: Processes, execution threads, synchronization constraints.
- **Call Structure**: Call relationships between processes.
- **Physical Structure**: Deployment of Data and Functions across computers, and across geographic sites.
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

System Communication
- Hardware infrastructure, communication medium.
- Synchronous/Asynchronous communication.
- Shared data, duplicated shared data.
- Parameter passing, global variables.
- Message passing.
- Routing, packet switching.
Software Architecture and Architectural Styles

System Control
- Procedure Call
  - Remote Procedure Call,
- Concurrent, Sequential,
- Event Based,
- Interrupt Based
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

**Architectural Styles:** Architecture Families, defined by:
- Component Types,
- Connector Types,
- Communication Protocols,
- Semantic Constraints.
Software Architecture and Architectural Styles

Architectural Styles: Different classifications,
- Independent Components, autonomous control, message communication.
- Event Based Systems, events trigger reactions (XL).
- Virtual Machines, layered architecture (old OS).
- Data Flow Architectures, availability of data as control (DP). Example: Pipe and Filter.
- Data Centered Architectures, shared data, communication medium (DBMS).
- Call and Return Architectures, single control thread (traditional PL code).
Software Architecture and Architectural Styles

Software architecture as a prerequisite for reuse:
- Because a prerequisite for CB development.
- Why are cars built from components (tires, batteries, engines, transmissions, horns, A/C, etc) and software products are not?
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

Software architecture as a prerequisite for reuse:
- Because a prerequisite for CB development.
- Why are cars built from components (tires, batteries, engines, transmissions, horns, A/C, etc) and software products are not?
- Because cars have had the same architecture for more than a century,
  - Software products don’t.
- To build a system from components, the provider of the component and the builder of the system must agree on a common architecture.
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

Pre-architecture software reuse experiments:
- Large software repositories,
- Sophisticated search algorithms,
- High quality software components,
- All paid for by the government, provided for free,
- Open for business.

What happens?
- Nothing, despite cost/risks of software.
  - It is not about functions, it is about architecture/architectural assumptions. No common architecture, no reuse.
SOFTWARE ARCHITECTURE AND ARCHITECTURAL STYLES

While software products in general do not have a common architecture, the elements of a product family may... Hence product line engineering.

- Most common form of successful reuse.
- E-commerce, compilers, banking, etc.
## Software Architecture and Architectural Styles

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Architectural Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Duplicate functions, fault tolerance.</td>
</tr>
<tr>
<td>Safety</td>
<td>Delegate safety critical functions to one component.</td>
</tr>
<tr>
<td>Security</td>
<td>Safeguard data, control access, multiply hoops.</td>
</tr>
<tr>
<td>Availability</td>
<td>Duplicate/distribute data, functions.</td>
</tr>
<tr>
<td>Throughput</td>
<td>Coarse grain components, minimize message traffic.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Fine grain components.</td>
</tr>
<tr>
<td>Response Time</td>
<td>Avoid bottlenecks, duplicate/distribute data, funs.</td>
</tr>
<tr>
<td>Portability</td>
<td>Isolate platform dependent components.</td>
</tr>
<tr>
<td>Failure Prob</td>
<td>Redundancy, fault removal.</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>No architectural impact.</td>
</tr>
<tr>
<td>Ease of Learning</td>
<td>Uniform interaction protocols.</td>
</tr>
<tr>
<td>Modularity</td>
<td>High cohesion, low coupling, contract programming.</td>
</tr>
</tbody>
</table>
PLAN

• Software Quality Attributes
• Software Architectures and Architectural Styles
• Requirements for An Architectural Description Language
• ACME and ACME+
  An Attribute Grammar for ACME+
  Compiling ACME+ Architectures
  Demo: Automated Architectural Analysis
• Conclusion: Summary and Prospects
• References
REQUIREMENTS FOR AN ARCHITECTURAL DESCRIPTION LANGUAGE

<table>
<thead>
<tr>
<th>Component</th>
<th>Response Time</th>
<th>Throughout</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Ra</td>
<td>Ta</td>
</tr>
<tr>
<td>B</td>
<td>Rb</td>
<td>Tb</td>
</tr>
</tbody>
</table>
Requirements for an Architectural Description Language

<table>
<thead>
<tr>
<th>Component</th>
<th>Response Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Ra</td>
<td>Ta</td>
</tr>
<tr>
<td>B</td>
<td>Rb</td>
<td>Tb</td>
</tr>
</tbody>
</table>

Attribute | Sequential Arch. | Parallel Arch. |
-----------|------------------|----------------|
Response Time | Ra+Rb            | Min(Ra, Rb)    |
Throughput    | Min(Ta, Tb)      | Ta+Tb          |
Requirements for an Architectural Description Language

Generalizing this type of reasoning for:
- Arbitrary topologies,
- Arbitrary quantitative attributes,
- Arbitrary number of links per component,
- Arbitrary relationships between links.
## Requirements for an Architectural Description Language

<table>
<thead>
<tr>
<th>ADL</th>
<th>Author</th>
<th>Institution</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>Garlan, Shaw</td>
<td>CMU</td>
<td>1995</td>
</tr>
<tr>
<td>AESOP</td>
<td>Garlan, Shaw</td>
<td>CMU</td>
<td>1994</td>
</tr>
<tr>
<td>Wright</td>
<td>Robert Allen</td>
<td>CMU</td>
<td>1997</td>
</tr>
<tr>
<td>MetaH</td>
<td>Vestal</td>
<td>Honeywell</td>
<td>1996</td>
</tr>
<tr>
<td>Lileanna</td>
<td>Tracz</td>
<td>IBM, Loral</td>
<td>1993</td>
</tr>
<tr>
<td>ArTek</td>
<td>London</td>
<td>Teknowledge</td>
<td>1995</td>
</tr>
<tr>
<td>C2</td>
<td>Taylor</td>
<td>UC Irvine</td>
<td>1996</td>
</tr>
<tr>
<td>Modechart</td>
<td>Al Mok</td>
<td>UT Austin</td>
<td>1996</td>
</tr>
</tbody>
</table>
# Requirements for an Architectural Description Language

<table>
<thead>
<tr>
<th>ADL</th>
<th>Author</th>
<th>Institution</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rapide</td>
<td>Luckham</td>
<td>Stanford</td>
<td>1995</td>
</tr>
<tr>
<td>UniCon</td>
<td>Shaw</td>
<td>CMU</td>
<td>1995</td>
</tr>
<tr>
<td>Darwin</td>
<td>Kramer</td>
<td>Imperial College</td>
<td>1995</td>
</tr>
<tr>
<td>SADL</td>
<td>Moriconi</td>
<td>SRI</td>
<td>1995</td>
</tr>
<tr>
<td>Demeter</td>
<td>Lieberherr</td>
<td>NEU</td>
<td>1996</td>
</tr>
<tr>
<td>CHAM</td>
<td>Berry</td>
<td>INRIA</td>
<td>1992</td>
</tr>
<tr>
<td>PSDL/CAPS</td>
<td>LuQi</td>
<td>NPS</td>
<td>1993</td>
</tr>
<tr>
<td>Resolve</td>
<td>Weide</td>
<td>Ohio State</td>
<td>1995</td>
</tr>
</tbody>
</table>
Requirements for an Architectural Description Language

- Support the ability to represent quantitative attributes of architectural components,
- Support a generic ontology of architectural components (adequate vocabulary),
- Support a rich vocabulary of functional dependencies to enable quantitative analysis (topology is insufficient),
- Automate analysis of system-level attributes from component-level attributes (compiling ADL to relevant equations).
REQUIREMENTS FOR AN ARCHITECTURAL DESCRIPTION LANGUAGE

To our surprise, none of the existing languages do that.

- *ACME* does represent quantitative attributes, but treats them as comments,
- *Wright* does represent operational information, but it is too detailed (yet at times fails to reflect the functional dependencies we are interested in), and it does not represent quantitative attributes,
- Etc.

Surprising given that architecture deals with quality / non-functional attributes.
Plan

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
  - An Attribute Grammar for ACME+
  - Compiling ACME+ Architectures
  - Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
ACME AND ACME+

ACME Ontology
- Components
  - Ports
- Connectors
  - Roles
- Systems
- Properties
- Representations
- Representation Map
ACME AND ACME+

ACME Ontology… Focus on:
- Components
  - Ports
- Connectors
  - Roles
- Systems
- Properties
- Representations
- Representation Map
## ACME AND ACME+

### Component
- Computational Element, or
- Data Store (stores and controls access to)
- Has ports, used to communicate with other components via connectors

### Connector
- Mediates interaction between components,
- Has roles, each defining a particular function in an interaction,
- Binary connectors: caller/callee; sender/receiver; reader/writer.
- N-ary connectors: broadcaster/receivers.
ACME and ACME+

system simple_cs = {
component client = {port sendRequest}; component server = {port receiveRequest}
connector rpc = {role {caller, callee}}
attachments : {client.sendRequest to rpc.caller; server.receiveRequest to rpc.callee}
ACME AND ACME+

ACME Properties

- Representing extra-structural attributes, in an open-ended manner,
- Arbitrary list of properties,
- Property, defined by: name, optional type, value.
- Any ACME entity may be annotated with a list of properties.
ACME AND ACME+

system simple_cs = {

component client = {port sendRequest;

    properties {requestRate: float = 17.0; sourceCode: file="code-lib/client.c" ;}}

component server = {port receiveRequest;

    properties {maxConcurrentClients: integer=1; multithreaded: Boolean=false;}}

connector rpc = {role {caller, callee};

    properties {synchronous:Boolean=true; maxRoles:integer=true;}}

attachments : {client.sendRequest to rpc.caller; server.receiveRequest to rpc.callee}
ACME AND ACME+

ACME+: Requirements,

- **Formal definition of properties**
  - From a selected catalog (response time, throughput, failure probability, MTTF, MTTR, etc),
  - Predefined units (seconds, transactions per second, probability scale, hours, hours, etc),
  - Relationships between ports of a component (input, output, duplicate, complementary, synchronous, asynchronous),
  - Relationships between roles of a connector (input, output, duplicate, complementary, synchronous, asynchronous, blocking, non-blocking, etc).
ACME AND ACME+

Relationships between ports of a component, relationships between roles of a connector.

- At a minimum, we wish to specify which are input ports and which are output ports.
- Among input ports, we wish to specify how they related to each other in providing input information.
  - input(anyof(P1,P2,P3)).
  - input(allof(P1,P2,P3)).
  - input(mostof(P1,P2,P3)).
ACME and ACME+

When more than one port is involved (as in all of, or most of) we must also specify whether they are synchronous or asynchronous:

- input(anyof(P1,P2,P3)).
- input(allof(asynch(P1,P2,P3))).
- input(mostof(asynch(P1,P2,P3))).
- input(allof(synchro(P1,P2,P3))).
- input(mostof(synchro(P1,P2,P3))).
ACME AND ACME+

As for output ports, in addition to designating them as such, we also need to determine the degree of overlap between them (duplicate, exclusive, overlap), as well as the synchronization between them (simultaneous, as available).

- `output(duplicate(P4,P5)).`
- `output(overlap(as available(P4,P5))).`
- `output(exclusive(simultaneous(P4,P5))).`
- `output(exclusive(as available(P4,P5))).`
ACME AND ACME+

Specified as part of a `fundep` statement (functional dependency), that specifies the relation between ports, and associates properties to the relation.

- `fundep` `{three2two
  input(mostof(synchro(P1,P2,P3))),
  output(exclusive(asavailable(P4,P5))),
  properties(processingTime=0.02,
    throughput=45,
    bufferCapacity=256,
    mtot=2000)}.`
ACME AND ACME+

Adequacy of the notation, availability of needed information:

- Aegis Weapons System,
  - Allen, CMU, Pittsburg, US
- Video Animation Repairing System,
  - Bonta, Bernardo, Univ di Bologna, Padova, Italy
- E-Bay e-commerce platform,
  - Ahmed, McGill University, Montreal, Canada
- Rule Based System.
  - Garlan and Shaw, Pittsburg, US
Plan

• Software Quality Attributes
• Software Architectures and Architectural Styles
• Requirements for An Architectural Description Language
• ACME and ACME+
• An Attribute Grammar for ACME+
• Compiling ACME+ Architectures
• Demo: Automated Architectural Analysis
• Conclusion: Summary and Prospects
• References
AN ATTRIBUTE GRAMMAR FOR ACME+

We consider canonical architectures as architectures that satisfy the following conditions:

- A single component with no input port and a single output port: the *source*.
- A single component with no output port and a single input port: the *sink*.
- Each component has designated input ports and output ports.
- Each connector has designated origin roles and destination roles.
- Each component/connector has relevant *fundep* relations, with relevant associated attributes.
AN ATTRIBUTE GRAMMAR FOR ACME+

Given a canonical architecture, we associate attributes (in the sense of attribute grammars) to each port and each role.

- These attributes represent the characteristics of a computation that initiates at the source component and reaches the port or role of interest. These attributes are synthesized from component and connector properties in light of:
  - Functional dependencies of components and connectors upstream of the port/role.
  - Component and connector properties associated with these functional dependencies.
AN ATTRIBUTE GRAMMAR FOR ACME+

Distinction between three concepts:

- **System level properties**: response time, throughput, MTTF, MTTR, etc. We represent these with upper case letters.

- **Component/connector-level properties**, which are described by ACME+’s *properties* construct, and are attached to components and connectors.

- **Port and role attributes** that we define in the sense of *attribute* grammars.
  - There is no one-to-one correspondence between properties and attributes.
AN ATTRIBUTE GRAMMAR FOR ACME+

While there is no one-to-one correspondence between component/connector properties and port/role attributes, there is one between port/role attributes and system level properties.

<table>
<thead>
<tr>
<th>Port/ Role Attributes</th>
<th>System level Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT</td>
<td>Response Time</td>
</tr>
<tr>
<td>TP</td>
<td>ThroughPut</td>
</tr>
<tr>
<td>FP</td>
<td>Failure Probability</td>
</tr>
<tr>
<td>MTTR</td>
<td>MeanTimeToRepair</td>
</tr>
<tr>
<td>MTTF</td>
<td>MeanTimeToFailure</td>
</tr>
</tbody>
</table>
AN ATTRIBUTE GRAMMAR FOR ACME+

Base Attribute Values
- source.outPort.RT = 0;
- source.outPort.TP=∞;
- source.outPort.FP=0;
- source.outPort.MTTF= ∞;

Attribute Values at Sink
- sink.inPort.RT = system.ResponseTime;
- sink.inPort.TP = system.Throughput;
- sink.inPort.FP = system.FailureProb;
- sink.inPort.MTTF = system.MeanTimeToFail.
An Attribute Grammar for ACME+

How do we propagate the attribute values from the source to the sink?

Inductive rules:

- Attachment statements: simple equations.
  - attachment {C.outPort to N.originRole}
  - We generate:
    - C.outPort.RT = N.originRole.RT,
    - C.outPort.TP = N.originRole.TP,
    - C.outPort.FP = N.originRole.FP,
    - C.outPort.MTTF = N.originRole.MTTF,
    - Etc...
An Attribute Grammar for ACME+

Propagation within components/ connectors (RT):

- Component, Single input port, single output port.
  - \( C.\text{outPort.RT} = C.\text{inPort.RT} + C.\text{ProcessingTime}. \)

- Connector, Single origin role, single destination role.
  - \( N.\text{destinationRole.RT} = N.\text{originRole.RT} + N.\text{TransmissionTime}. \)
AN ATTRIBUTE GRAMMAR FOR ACME+

Multiple ports, roles (component, RT):

- fundep {three2two,
  
  input(allof(asynch(P1,P2,P3))),
  output(P4),

  properties {processingTime= , ...}}

- C.P4.RT = C.ProcessingTime +
  
An Attribute Grammar for ACME+

Multiple ports, roles (component, RT):

- fundep {three2two,
  
  input(anyof(asynch(P1,P2,P3))),
  output(P4),
  properties {processingTime= , ...}}

- C.P4.RT = C.ProcessingTime +
AN ATTRIBUTE GRAMMAR FOR ACME+

Multiple ports, roles (component, RT):

- `fundep {three2two,
  input(mostof(asynch(P1,P2,P3))),
  output(P4),
  properties {processingTime= , ...}}`

- `C.P4.RT = C.ProcessingTime +
AN ATTRIBUTE GRAMMAR FOR ACME+

Multiple ports, roles (component, RT):

- fundep {three2two,
  input(xxxxx(synchro(P1,P2,P3))),
  output(P4),
  properties {processingTime=, ...}}

- C.P4.RT = C.ProcessingTime +
PLAN

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
  - An Attribute Grammar for ACME+
  - Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
Compiling ACME+ Architectures

ACME+:
- ACME
  - ACME-style properties
  + fundep statements, along with their properties.

To generate ACME+ architectures:
- We use ACME Studio to generate plain architecture,
- Check its syntax, validate it using GUI,
- Add fundep statements,
  - Run ACME+ compiler.
- Output: a set of equations linking all attribute values, from source to sink, and sink values to system properties.
COMPILED ACME+ ARCHITECTURES

Available Functionalities,

- Computer system level Properties from component/connector level properties.
- Produces explicit expression of system level Properties from component/connector level properties.
- For some system level Properties, identifies the component/connector that causes the bottleneck with respect to that property.
PLAN

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
  An Attribute Grammar for ACME+
- Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
DEMO: AUTOMATED ARCHITECTURAL ANALYSIS

Aegis Weapons System
PLAN

• Software Quality Attributes
• Software Architectures and Architectural Styles
• Requirements for An Architectural Description Language
• ACME and ACME+
• An Attribute Grammar for ACME+
• Compiling ACME+ Architectures
• Demo: Automated Architectural Analysis
• Conclusion: Summary and Prospects
• References
CONCLUSION: SUMMARY AND PROSPECTS

Summary

- Software qualities cannot be achieved simultaneously, and must be prioritized.

- Software Architecture represent early design decisions that affect quality attributes of the final product.
  - Hence the architecture of a software product must be designed according to the selected quality attributes.

- A proliferation of ADL, mostly during the nineties.

- Contribution: an ACME extension and support tool that enable us to reason about quality attributes.
CONCLUSION: SUMMARY AND PROSPECTS

Prospects

- Extend to non canonical architectures (cyclic topologies, concurrent processing),
- Other Quality Attributes,
- User Defined Quality Attributes,
- Broaden the analysis of architectures, notably by means of the explicit expression of system level properties.
Plan

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
- An Attribute Grammar for ACME+
- Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References
REFERENCES

Plan

- Software Quality Attributes
- Software Architectures and Architectural Styles
- Requirements for An Architectural Description Language
- ACME and ACME+
- An Attribute Grammar for ACME+
- Compiling ACME+ Architectures
- Demo: Automated Architectural Analysis
- Conclusion: Summary and Prospects
- References