



Methodical Design of Software Architecture Using an Architecture Design Assistant (ArchE)

Felix Bachmann and Mark Klein
Software Engineering Institute

Sponsored by the U.S. Department of Defense
© 2005 by Carnegie Mellon University

Version 1.0

page 1



Outline

Motivation

Principles

ArchE

Example

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE APR 2005		2. REPORT TYPE		3. DATES COVERED 00-00-2005 to 00-00-2005	
4. TITLE AND SUBTITLE Methodical Design of Software Architecture Using an Architecture Design Assistant (ArchE)				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Carnegie Mellon University,Software Engineering Institute,Pittsburgh,PA,15213				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 23	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			



The Key Question



How do we systematically move from a set of requirements to a software architecture that satisfies those requirements?



The Problem

Designing is very knowledge intensive:

- The required expertise rarely resides in one place/person
- It's unclear how/what knowledge should drive design

Knowledge requirements:

- Domain
- Quality attribute (e.g. *performance, security, modifiability*)
- Architectural design
- Design methodology
-



Our Goals

Goal: To methodically design software architectures so that they predictably meet quality attribute requirements.

Sub-goals:

- Determine/discover fundamental design principles
- Operationalize principles via method(s) (“Attribute Driven Design”)
- Investigate techniques and build prototypes for automated support (ArchE)



Outline

Motivation

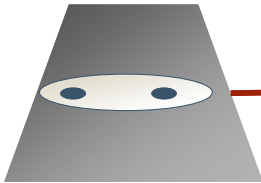
Principles

ArchE

Example



Types of Requirements

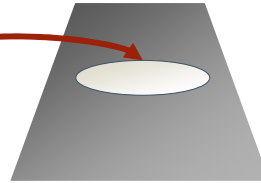


Requirements

Constraints – pre-specified design decisions

Features – what functions add value to the user (e.g. what the system does)

Quality Attribute– how well the system does by various measures (e.g., how timely, secure, modifiable it is)



Software Architectures



What type of requirements drive architectural design?

Answer: Functional requirements are least important for architecture design – quality requirements and constraints are most important

Here's some evidence:

If the only concern is functionality then a monolithic system would suffice.

However it is quite common to see:

- Redundancy structures for reliability
- Concurrency structures for performance
- Layers for modifiability



What does an architect/ArchE need to know to methodically design?

Knowledge requirements

- Quality knowledge – how to achieve required qualities in an architecture design
- Architecture design process – how to get an architecture from requirements

Our approach:

- Precisely define quality attribute requirements in terms of scenarios.
- Exploit the “structure” of quality attribute models to define the structure of well-formed architectures.
- Define transformations between architecture models, quality attribute models, quality attribute scenarios and quality attribute measures.



We have a common form for specification of quality requirements

We use **quality attribute general scenarios**, which are system independent, to guide the specification of quality attribute requirements.

We characterize quality attribute requirements for a specific system by a collection of **concrete quality attribute scenarios**. These are instances of general scenarios.

We use **general scenario generation tables** to construct well-formed general scenarios for each attribute.



General Scenarios

General scenarios have six parts. The “values” for each part define a vocabulary for articulating quality attribute requirements. The parts are:

- Stimulus
- Source of stimulus
- Environment in which the stimulus arrives
- Artifact influenced by the stimulus
- Response of the system to the stimulus
- Response measures



Availability Scenario Generation Table

Source of stimulus:

- Internal to the system
- ✓ External to the system

Environment:

- ✓ Normal operation
- Degraded mode

Response:

- ✓ record it
- ✓ notify parties
- ✓ operate in normal or degraded mode

Example Scenario:

“An unanticipated message is received by a system process during normal operation. The process has to record it, inform the appropriate parties and continue to operate in normal mode without any downtime.”

Stimulus:

- ✓ Unanticipated event
- Update to a data store

Artifact:

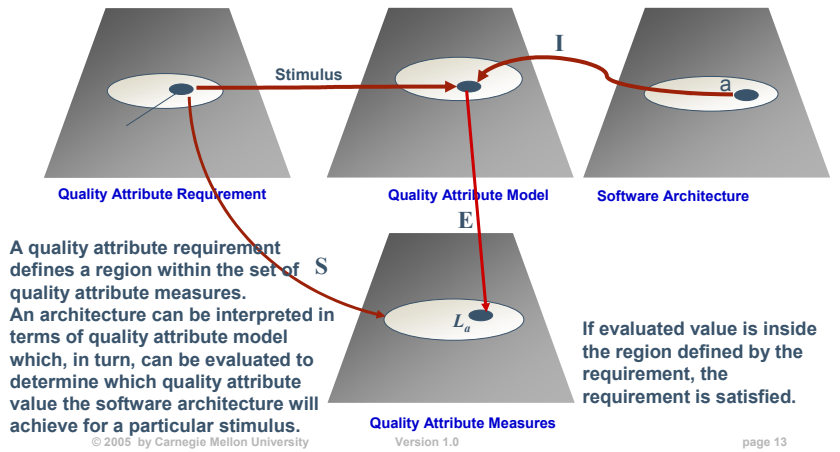
- ✓ Process
- Persistent storage

Response measures:

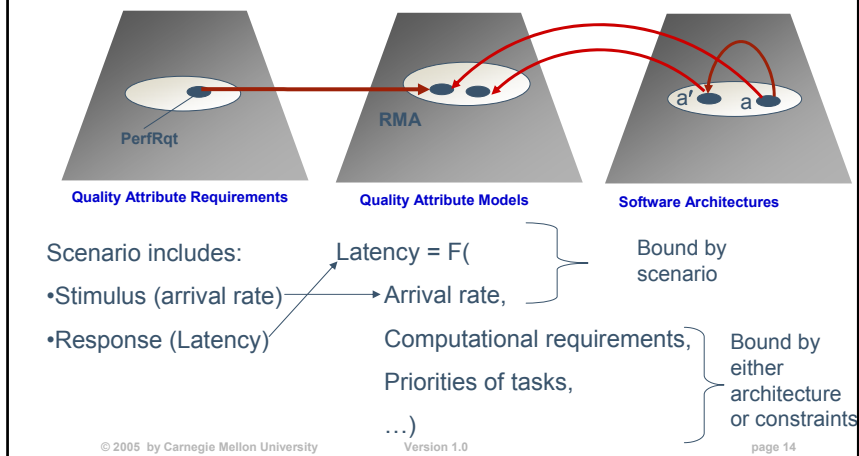
- ✓ Availability percentage
- Time range in which the system can be in degraded mode



What does it mean to satisfy a quality attribute requirement?

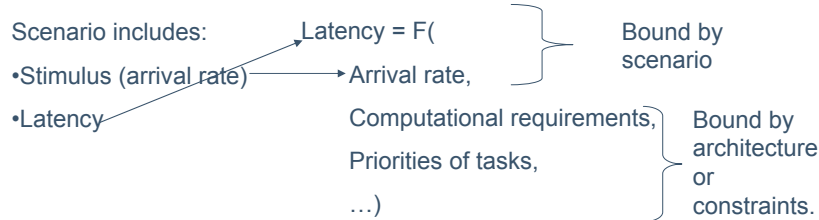


Quality Attribute Models





Parameters define architectural tactics



Tactics are designed to adjust the parameters.

Can work backwards – determine which values of parameters will satisfy latency, with given arrival rate, and then ask whether these values are architecturally achievable using tactics.

May also weaken constraints or requirements using tactics.

© 2005 by Carnegie Mellon University

Version 1.0

page 15



What are architectural tactics?

For the six quality attributes –availability, modifiability, performance, security, testability, usability - we have enumerated a collection of “tactics”

Formal definition: An *architectural tactic* is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural design decisions.

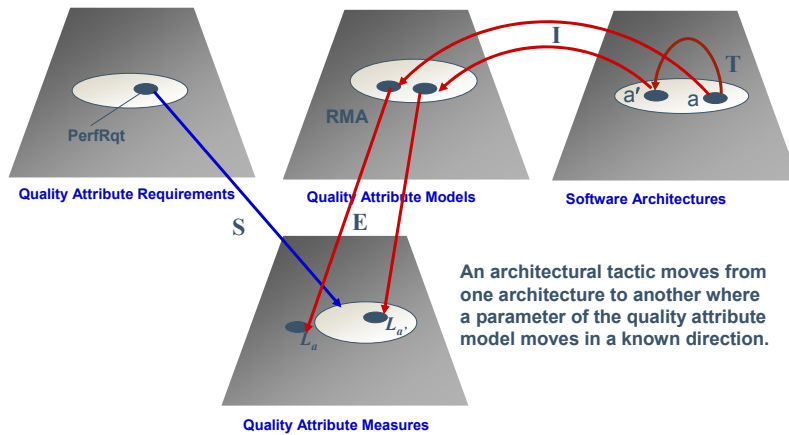
© 2005 by Carnegie Mellon University

Version 1.0

page 16



Architectural Tactics



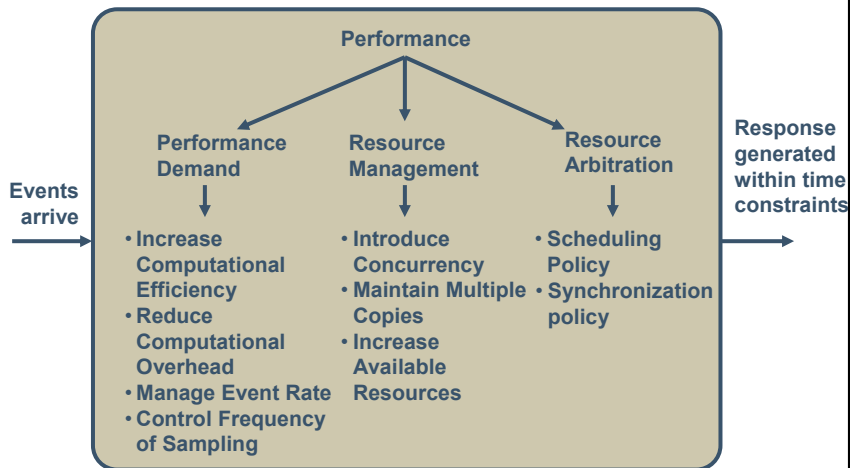
© 2005 by Carnegie Mellon University

Version 1.0

page 17



Performance Tactics



© 2005 by Carnegie Mellon University

Version 1.0

page 18



Outline

Motivation

Principles

ArchE

Example



ArchE – Architectural Expert

ArchE is a tool intended to complement an architect during the design process

Our vision is that

- The architect has domain knowledge and an understanding of what is feasible
- ArchE has knowledge of quality attributes and their relation to design

ArchE is emerging work at the SEI.



ArchE vis a vis any particular quality attribute

Quality attribute theories are created and change over time

We want ArchE infrastructure to be independent of any particular quality attribute

- ArchE is modular with respect to quality attributes that are included
- We use term “reasoning framework” to describe how quality attribute knowledge is encapsulated in ArchE.
- We view reasoning frameworks as “plug-ins”



Process of using ArchE (current version)

Architect: provide scenarios and features to ArchE

ArchE: generates initial architecture based on reasoning frameworks and scenarios

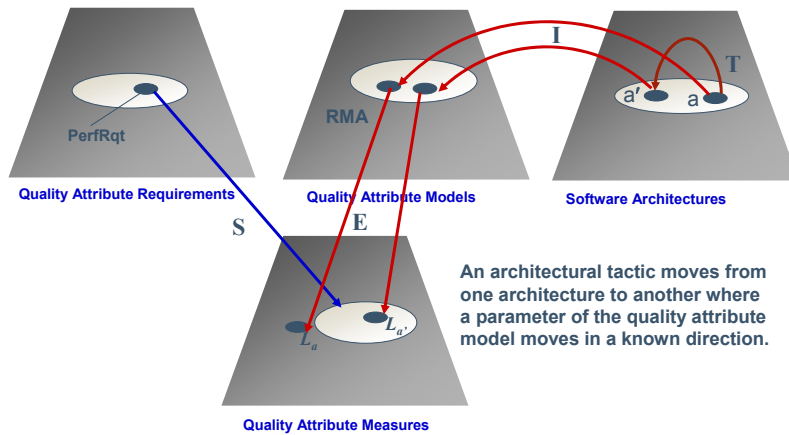
ArchE: presents list of possible tactics to improve architecture to architect

Architect: choose tactic to apply

ArchE: apply tactic and generate new list of possible tactics



ArchE Uses Tactics to Move Architecture in the Design Space



© 2005 by Carnegie Mellon University

Version 1.0

page 23



Outline

Motivation

Principles

ArchE

Example

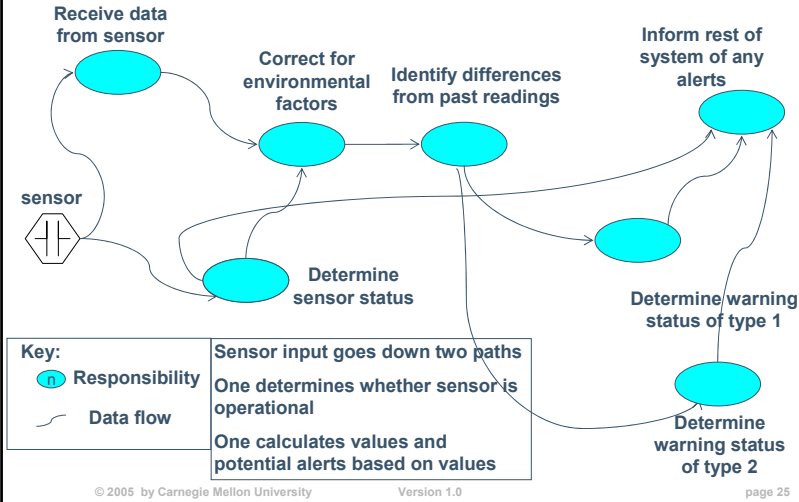
© 2005 by Carnegie Mellon University

Version 1.0

page 24



Initial Functions for Sensor Demo



Functions as they are entered into Arche

1. Receive data from sensor (Receive)
 2. Correct for environmental factors (Correct)
 3. Determine sensor status (Status)
 4. Identify warning conditions (Detect)
 - 4.1 Identify differences from past readings (Diff)
 - 4.2 Determine warning status
 - 4.2.1 Determine warning status of type 1 (Type 1)
 - 4.2.2 Determine warning status of type 2 (Type 2)
 5. Inform rest of the system of any errors (Inform)
- (Demo step 1)**



Scenarios for the Sample Problem

Modifiability

1. Replace sensor without change to functionality within 4 person days
2. Add new warning status without impacting existing warning statuses within 2.5 person days

Performance

1. Determine sensor status within 250 ms after receiving sensor input. Sensor input arrives every 500ms
2. Determine differences from past readings within 1250ms after receiving input. Input arrives every 1600ms.
3. Inform the rest of system of any alerts within 350ms after the arrival of alert status. Alert status arrives every 350ms.

(Demo step 2)



Relate Scenarios to Responsibilities

Responsibilities and relations among responsibilities carry parameters.

Scenarios are not yet related to responsibilities.

Costs, execution times, and dependency are not yet assigned

Thus, there is not enough information for ArchE to determine whether the scenarios can be met.



Initial Architecture for Impact Analysis

If no assignment of responsibilities to modules then assign each responsibility from initial set to its own module.

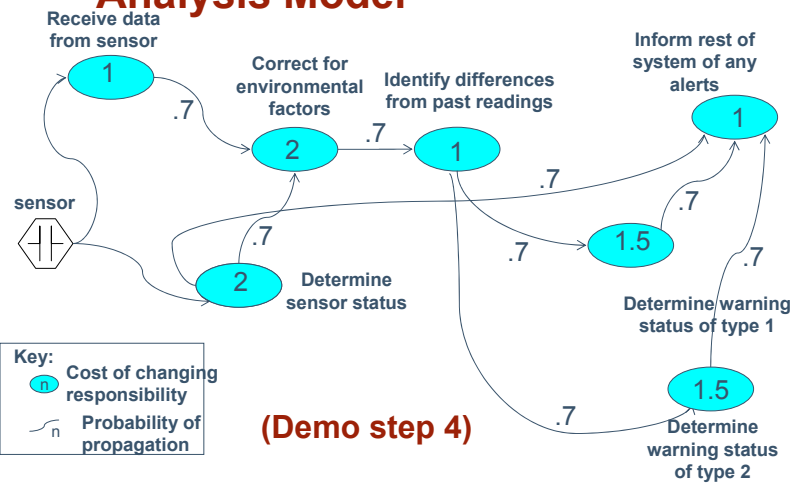
(Demo step 3)

Retrieve parameters from architect.

- Cost of change of responsibility
- Probability of change propagating



Parameterized Values of Impact Analysis Model



(Demo step 4)



Scenarios for the Sample Problem

Modifiability

1. **Replace sensor without change to functionality within 4 person days**
2. Add new warning status without impacting existing warning statuses within 2.5 person days

Performance

1. Determine sensor status within 250 ms after receiving sensor input. Sensor input arrives every 500ms
2. Determine differences from past readings within 1250ms after receiving input. Input arrives every 1250ms.
3. Inform the rest of system of any alerts within 350ms after the arrival of alert status. Alert status arrives every 350ms.

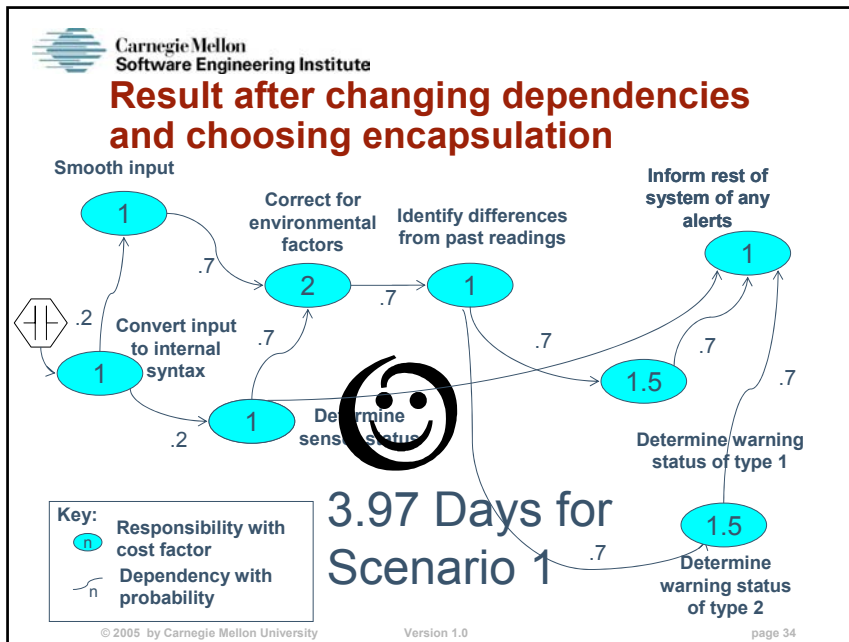
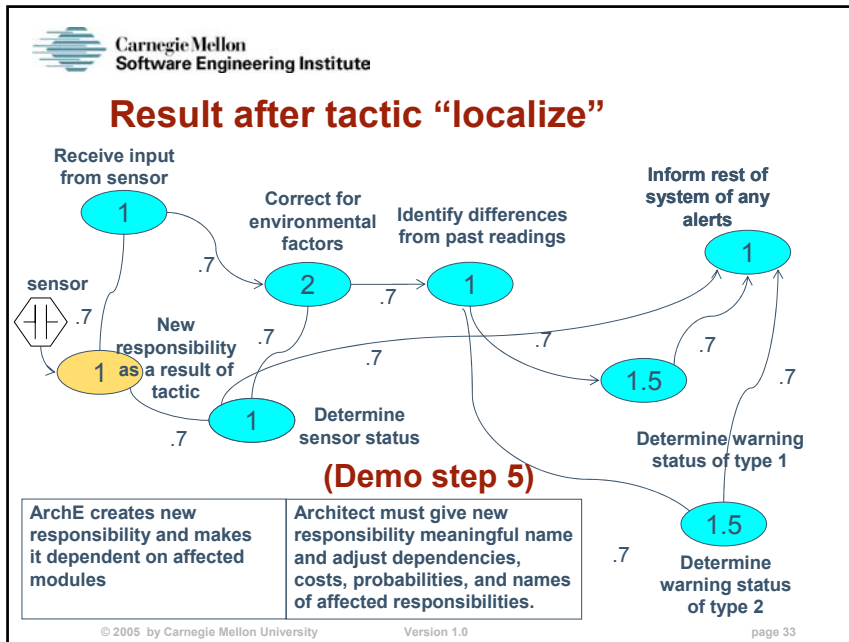


ArchE Proposes Possible Tactics

For modifiability ArchE can propose tactics like:

- Localization
- Encapsulation
- wrappers

We choose “localize”





Scenarios for the Sample Problem

Modifiability

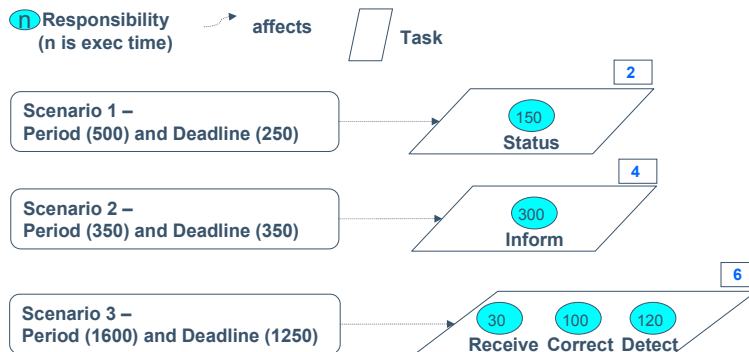
1. Replace sensor without change to functionality within 3 person days
2. Add new warning status without impacting existing warning statuses within 2 person days

Performance

1. Determine sensor status within 250 ms after receiving sensor input. Sensor input arrives every 500ms
2. Determine differences from past readings within 1250ms after receiving input. Input arrives every 1600ms.
3. Inform the rest of system of any alerts within 350ms after the arrival of alert status. Alert status arrives every 350ms.



Initial Architecture



Create a task for each scenario.
Assign deadline monotonic priorities to the tasks



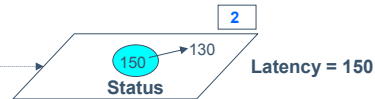
Evaluate Model

 Responsibility
(n is exec time)

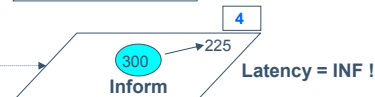
→ affects

 Task

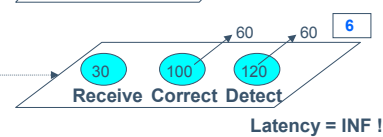
Scenario 1 –
Period (500) and Deadline (250)



Scenario 2 –
Period (350) and Deadline (350)



Scenario 3 –
Period (1600) and Deadline (1250)



Total utilization > 1.0 and deadlines are violated !

Tactic: Try reducing execution times of several
responsibilities

(Demo step 6)



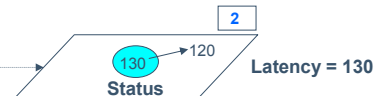
Applying Tactics -1

 Responsibility
(n is exec time)

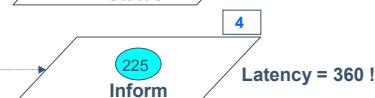
→ affects

 Task

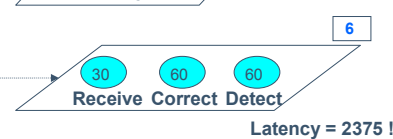
Scenario 1 –
Period (500) and Deadline (250)



Scenario 2 –
Period (350) and Deadline (350)



Scenario 3 –
Period (1600) and Deadline (1250)



Total utilization < 1.0 but deadlines are still violated !

Tactic: Try reducing execution time of *Status*.

(Demo step 7)



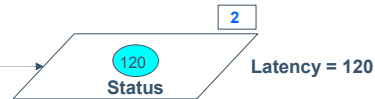
Applying Tactics -2

 Responsibility
(n is exec time)

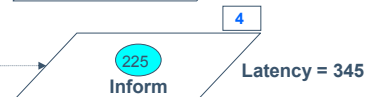
→ affects

 Task

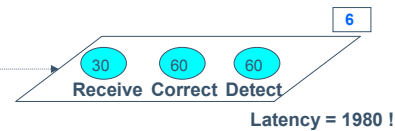
Scenario 1 –
Period (500) and Deadline (250)



Scenario 2 –
Period (350) and Deadline (350)



Scenario 3 –
Period (1600) and Deadline (1250)



One deadline is still violated !
Tactic: Try increasing period of *Inform*.

(Demo step 8)



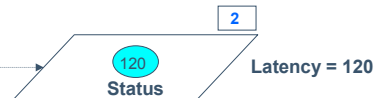
Applying Tactics -3

 Responsibility
(n is exec time)

→ affects

 Task

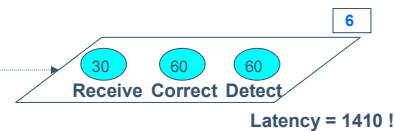
Scenario 1 –
Period (500) and Deadline (250)



Scenario 2 –
Period (385) and Deadline (350)



Scenario 3 –
Period (1600) and Deadline (1250)



One deadline is still violated !
Tactic: Try increasing period of *Status*.



Applying Tactics -4

 Responsibility
(n is exec time)

→ affects

 Task

Scenario 1 –
Period (550) and Deadline (250)

120

Status

2

Latency = 120

Scenario 2 –
Period (385) and Deadline (350)

225

Inform

4

Latency = 345

Scenario 3 –
Period (1600) and Deadline (1250)

30

60

60

Receive Correct Detect

6

Latency = 1065

*** Success ***

(Demo step 8)



Status

Applying ArchE to realistic examples

- ArchE has demonstrated that methodical design with predictable results is possible for small systems.
- We are looking for collaborators to help us with the extension of PAD and ArchE.

Extensions to ArchE that are underway

- Input constraints
- ArchE proposes patterns as well as tactics
- Variability reasoning framework
- Extension of performance reasoning framework



Future Work - 1

Make searching more efficient

- Patterns presented to architect as well as tactics
- Tradeoffs managed in a better fashion
- Better initial guess at architecture
- More sophisticated search
- Learning based on past choices



Future Work - 2

Make more and better reasoning frameworks

- More depth in current reasoning frameworks
- Add reasoning frameworks for other attributes (e.g., variability, security, dependability)
- Develop domain specific language for specification of reasoning frameworks
- Make ArchE more realistic
 - Apply to more sophisticated problems
 - Improve the user interface



More Information

Three SEI technical reports available on our web site:

1. *Illuminating the fundamental contributors to software architecture quality*. CMU/SEI-2002-TR-025
2. *Deriving architectural tactics: A step toward methodical architectural design*. CMU/SEI-2003-TR-004
3. *Preliminary Design of ArchE: A Software Architecture Design Assistant*. CMU/SEI-2003-TR-021

Lists of general scenarios and tactics are available in second edition of *Software Architecture in Practice*

