Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

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

Carnegie Mellon
Software Engineering Institute

Outline

Motivation

Principles

ArchE

Example

Carnegie Mellon
Software Engineering Institute

The Key Question

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

Carnegie Mellon
Software Engineering Institute

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
- ….

Carnegie Mellon
Software Engineering Institute

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)

Carnegie Mellon
Software Engineering Institute

Outline

Motivation

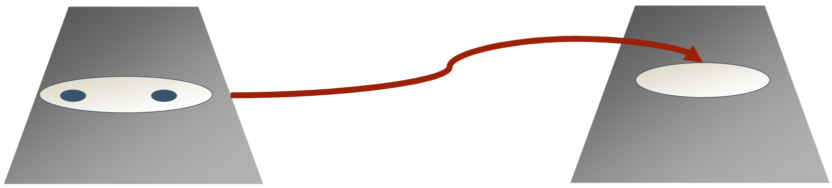
Principles

ArchE

Example

Carnegie Mellon
Software Engineering Institute

Types of Requirements



Requirements **Software Architectures**

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)

 Version 1.0 page 7

Carnegie Mellon
Software Engineering Institute

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 is it quite common to see:

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

 Version 1.0 page 8

Carnegie Mellon
Software Engineering Institute

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.

Carnegie Mellon
Software Engineering Institute

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.

Carnegie Mellon
Software Engineering Institute

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

Carnegie Mellon
Software Engineering Institute

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

Stimulus:
- ✓ Unanticipated event
- Update to a data store

Artifact:
- ✓ Process
- Persistent storage

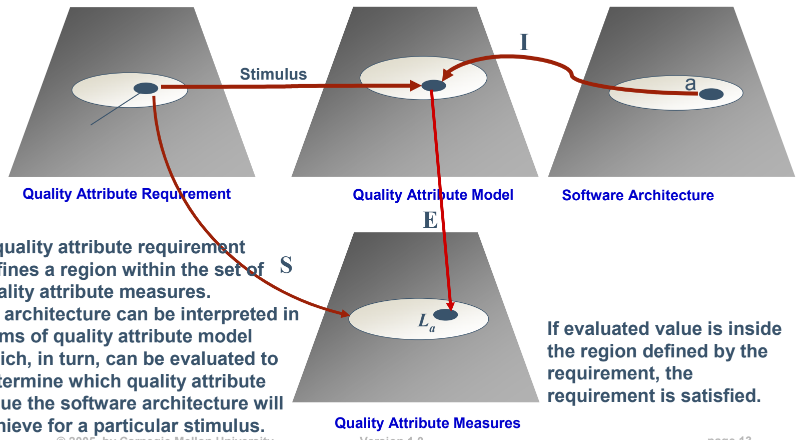
Response measures:
- ✓ Availability percentage
- Time range in which the system can be in degraded mode

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."

Carnegie Mellon
Software Engineering Institute

What does it mean to satisfy a quality attribute requirement?



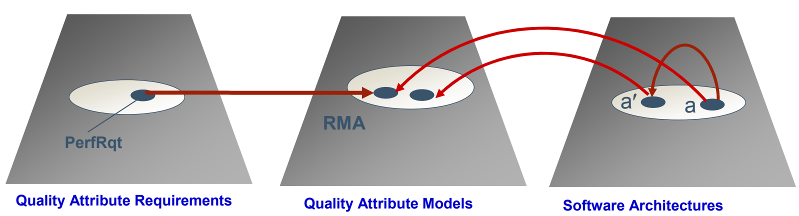
A quality attribute requirement defines a region within the set of quality attribute measures.
An architecture can be interpreted in terms of quality attribute model which, in turn, can be evaluated to determine which quality attribute value the software architecture will achieve for a particular stimulus.

If evaluated value is inside the region defined by the requirement, the requirement is satisfied.

© 2005 by Carnegie Mellon University Version 1.0 page 13

Carnegie Mellon
Software Engineering Institute

Quality Attribute Models



Scenario includes:

•Stimulus (arrival rate)

•Response (Latency)

Latency = F(

Arrival rate,

} Bound by scenario

Computational requirements,

Priorities of tasks,

…)

} Bound by either architecture or constraints

© 2005 by Carnegie Mellon University Version 1.0 page 14

Carnegie Mellon
Software Engineering Institute

Parameters define architectural tactics

Scenario includes:

•Stimulus (arrival rate)

•Latency

Latency = F(
Arrival rate,
Computational requirements,
Priorities of tasks,
…)

Bound by scenario (Latency, Arrival rate)

Bound by architecture or constraints. (Computational requirements, Priorities of tasks, …)

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.

 Version 1.0

Carnegie Mellon
Software Engineering Institute

What are architectural tactics?

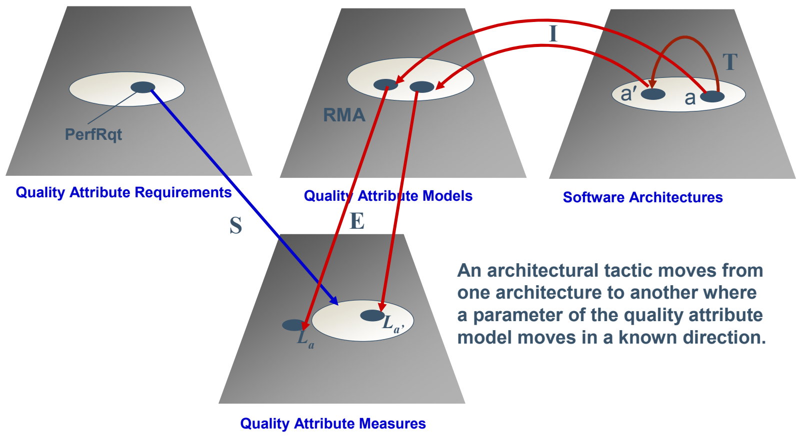
For the six quality attributes –availability, modifiability, performance, security, testability, usability - we have enumerated a collection of "tactics"

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.

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Architectural Tactics

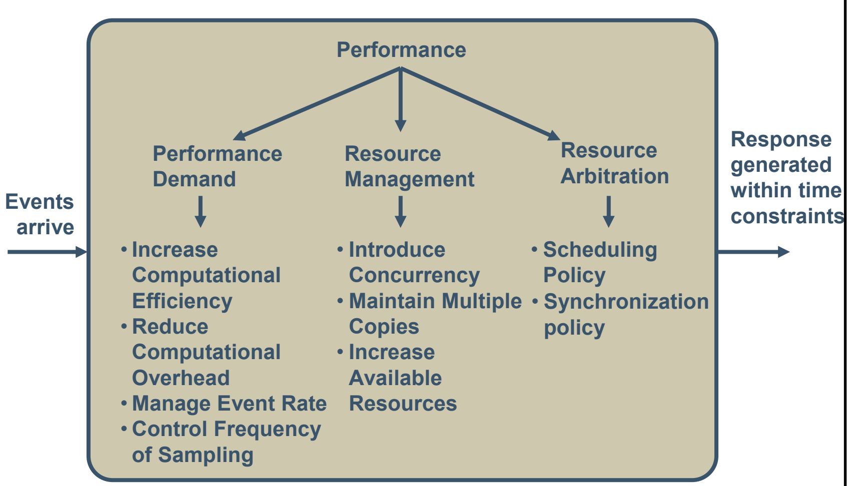


An architectural tactic moves from one architecture to another where a parameter of the quality attribute model moves in a known direction.

 Version 1.0 page 17

Carnegie Mellon
Software Engineering Institute

Performance Tactics



 Version 1.0 page 18

Carnegie Mellon
Software Engineering Institute

Outline

Motivation

Principles

ArchE

Example

Carnegie Mellon
Software Engineering Institute

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.

Carnegie Mellon
Software Engineering Institute

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"

 Version 1.0 page 21

Carnegie Mellon
Software Engineering Institute

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

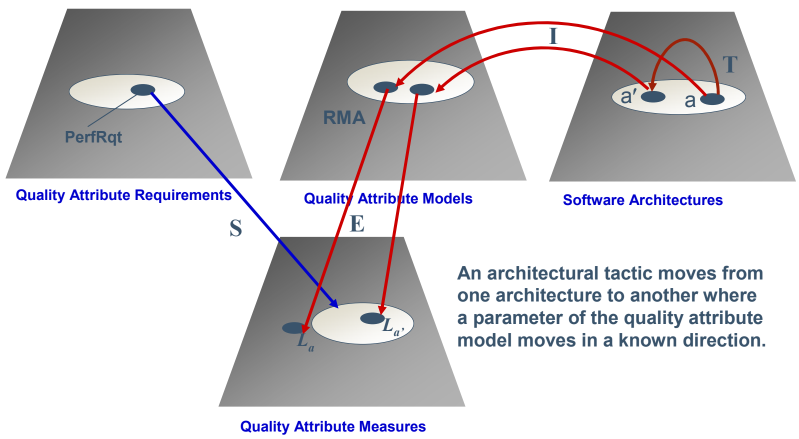
Architect: choose tactic to apply

ArchE: apply tactic and generate new list of possible tactics

 Version 1.0 page 22

Carnegie Mellon
Software Engineering Institute

ArchE Uses Tactics to Move Architecture in the Design Space



An architectural tactic moves from one architecture to another where a parameter of the quality attribute model moves in a known direction.

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Outline

Motivation

Principles

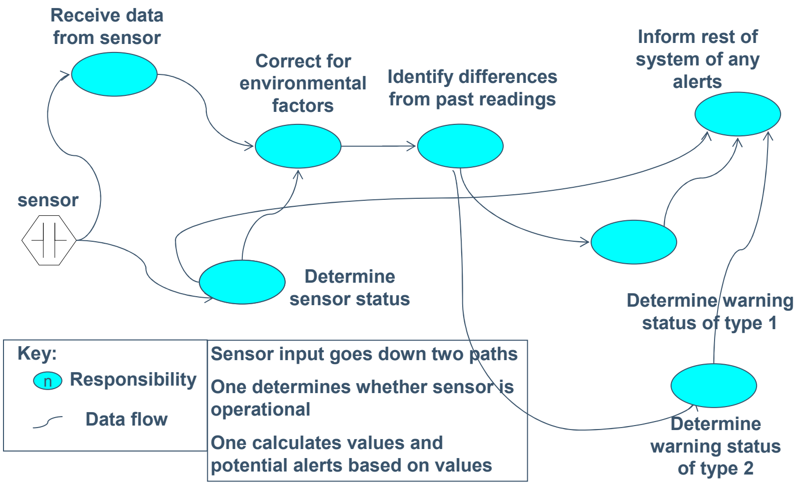
ArchE

Example

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Initial Functions for Sensor Demo



 Version 1.0 page 25

Carnegie Mellon
Software Engineering Institute

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)

 Version 1.0 page 26

Carnegie Mellon
Software Engineering Institute

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)

 Version 1.0

Carnegie Mellon
Software Engineering Institute

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.

 Version 1.0

Carnegie Mellon
Software Engineering Institute

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)

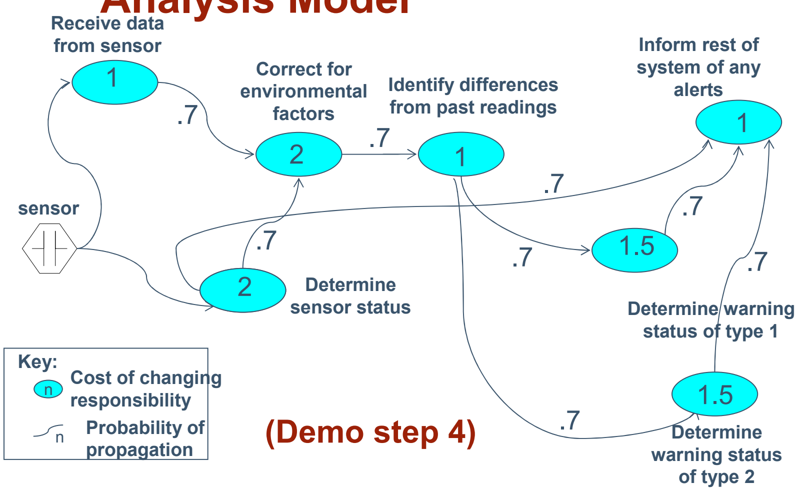
Retrieve parameters from architect.

- Cost of change of responsibility
- Probability of change propagating

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Parameterized Values of Impact Analysis Model



(Demo step 4)

 Version 1.0

Carnegie Mellon
Software Engineering Institute

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.

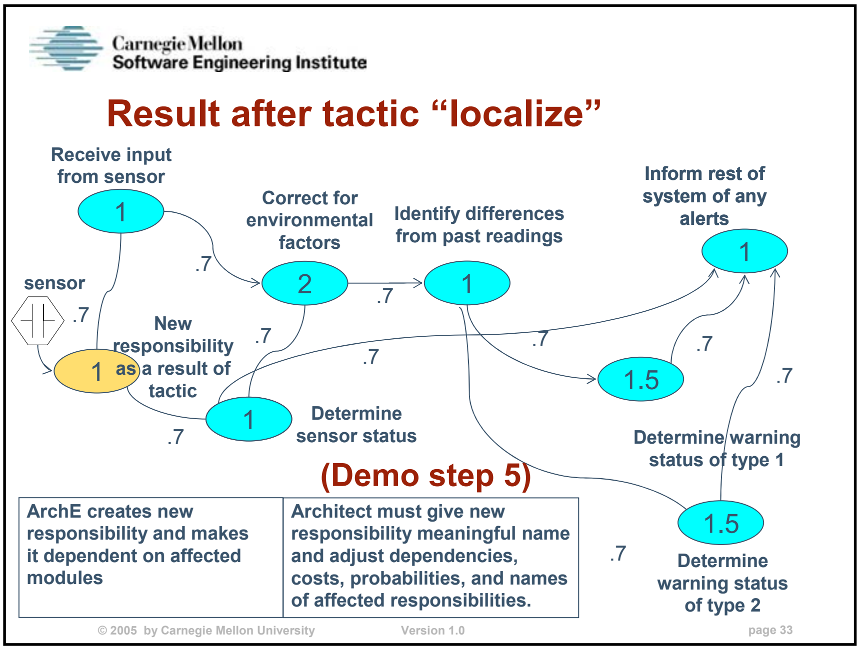
Carnegie Mellon
Software Engineering Institute

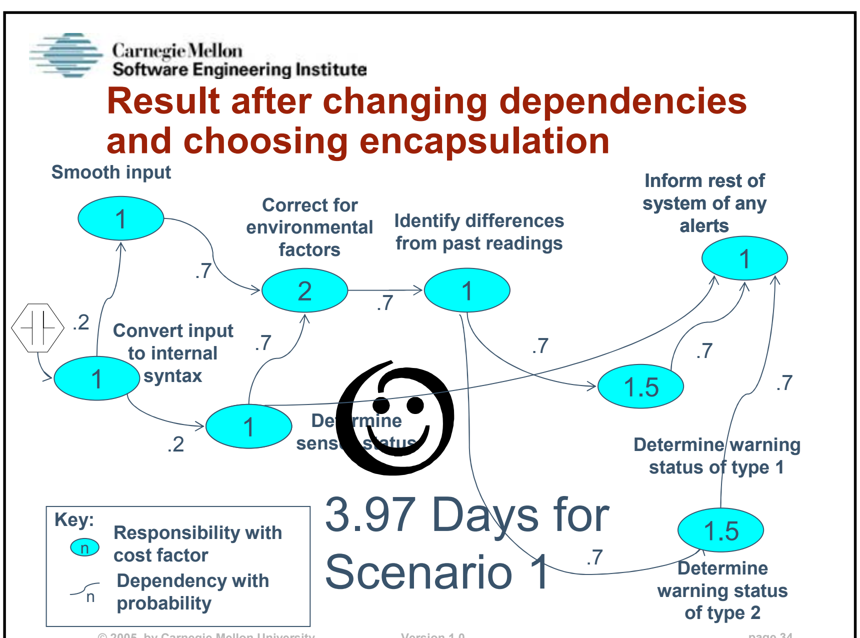
ArchE Proposes Possible Tactics

For modifiability ArchE can propose tactics like:

- Localization
- Encapsulation
- wrappers

We choose “localize”


Carnegie Mellon
Software Engineering Institute
Result after tactic "localize"
Receive input from sensor
1
Correct for environmental factors
2
Identify differences from past readings
1
Inform rest of system of any alerts
1
sensor
.7
.7
.7
.7
.7
.7
.7
.7
New responsibility as a result of tactic
1
1
Determine sensor status
1.5
Determine warning status of type 1
(Demo step 5)
ArchE creates new responsibility and makes it dependent on affected modules
Architect must give new responsibility meaningful name and adjust dependencies, costs, probabilities, and names of affected responsibilities.
1.5
.7
Determine warning status of type 2
© 2005 by Carnegie Mellon University
Version 1.0
page 33


Carnegie Mellon
Software Engineering Institute
Result after changing dependencies and choosing encapsulation
Smooth input
1
Correct for environmental factors
2
Identify differences from past readings
1
Inform rest of system of any alerts
1
.7
.7
.2
Convert input to internal syntax
.7
.7
.7
.7
1
1
Determine sensor status
1.5
.2
Determine warning status of type 1
Key:
Responsibility with cost factor
n
Dependency with probability
n
3.97 Days for Scenario 1
.7
1.5
Determine warning status of type 2
© 2005 by Carnegie Mellon University
Version 1.0
page 34

Carnegie Mellon
Software Engineering Institute

Scenarios for the Sample Problem

Modifiability

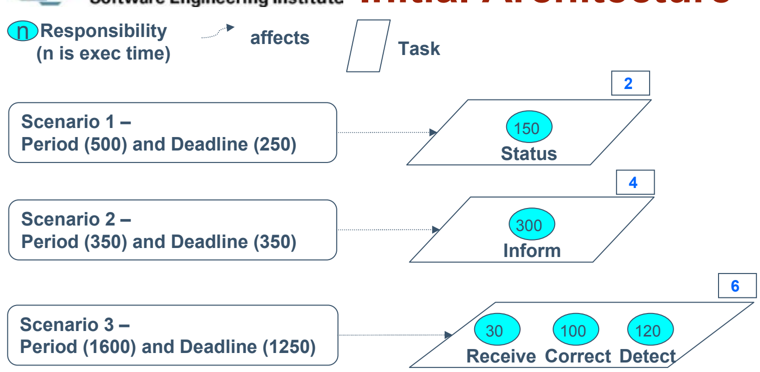
1. Replace sensor without change to functionality within 3 person days
2. Add new warning status without impacting existing warning statuses within 2 person days

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.**

Carnegie Mellon
Software Engineering Institute

Initial Architecture

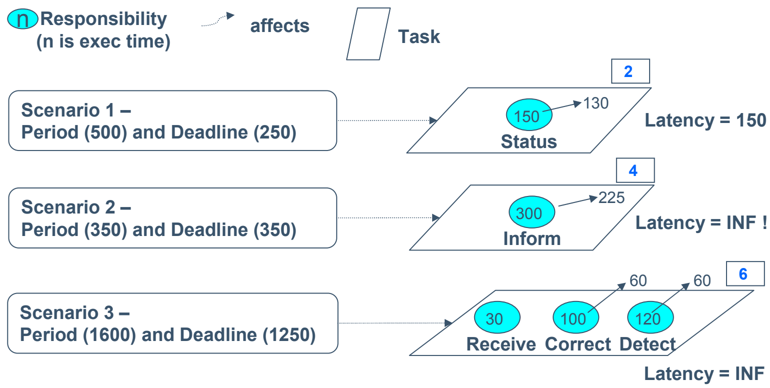


Create a task for each scenario.

Assign deadline monotonic priorities to the tasks

Carnegie Mellon
Software Engineering Institute

Evaluate Model

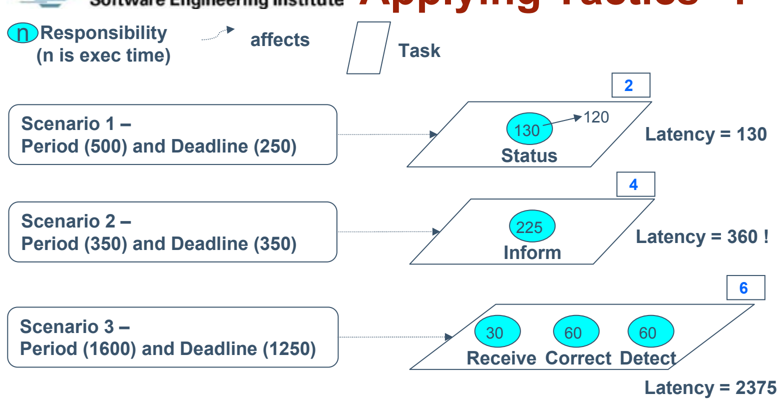


Total utilization > 1.0 and deadlines are violated !

Tactic: Try reducing execution times of several responsibilities **(Demo step 6)**

Carnegie Mellon
Software Engineering Institute

Applying Tactics -1

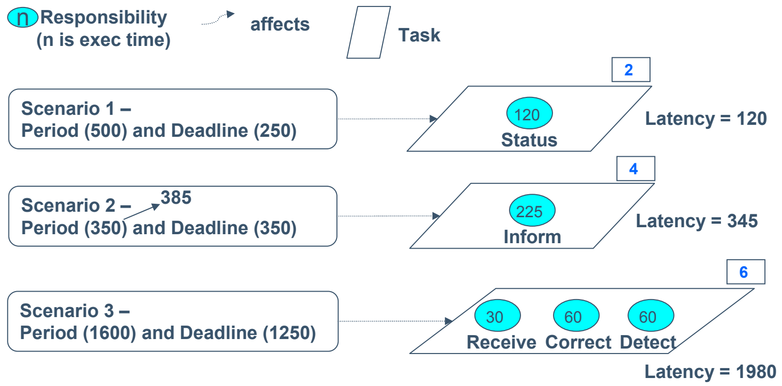


Total utilization < 1.0 but deadlines are still violated !

Tactic: Try reducing execution time of *Status*.

(Demo step 7)

Applying Tactics -2



One deadline is still violated !

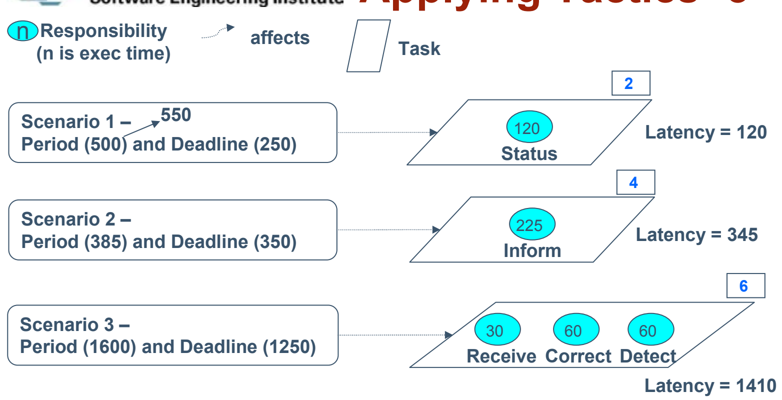
Tactic: Try increasing period of *Inform*.

(Demo step 8)

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Applying Tactics -3



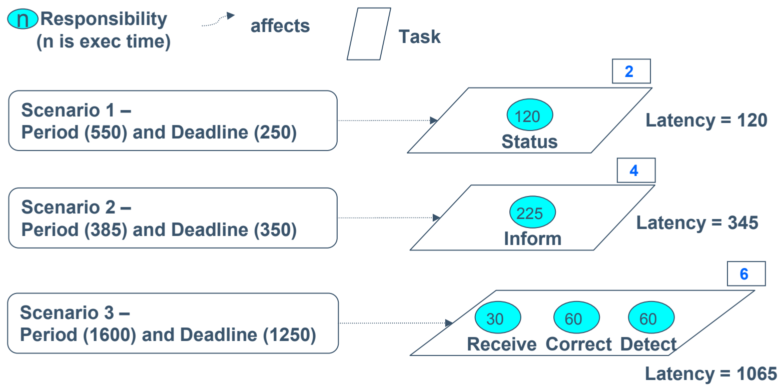
One deadline is still violated !

Tactic: Try increasing period of *Status*.

 Version 1.0

Carnegie Mellon
Software Engineering Institute

Applying Tactics -4



*** Success ***

(Demo step 8)

Carnegie Mellon
Software Engineering Institute

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

Carnegie Mellon
Software Engineering Institute

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

Carnegie Mellon
Software Engineering Institute

Future Work - 2

Make more and better reasoning frameworks

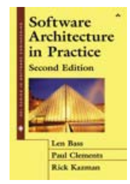
- More depth in current reasoning frameworks
- Add reasoning frameworks for other attributes (e.g., variability, security, dependability)
- Develop domain specific language for specification of reasoning frameworks
- Make ArchE more realistic
  - Apply to more sophisticated problems
  - Improve the user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



 Version 1.0