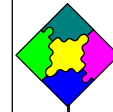




4. Analysing Style-based Software Architectures



Outline

- The importance of Predictability in Software Design
- The notion of Quality Attribute
- Attribute-based Architectural Styles (ABAS)
- Analysis support in ABAS

2



Building Complex Software

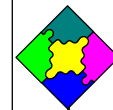
The complexity of software is continually growing:

- in scale
- in scope
- in requirements

The criticality and pervasiveness of software is continually growing.

Our *reliance* on software is ever increasing.

3



Engineering in Predictability

To build large complex systems you must have predictability. We can not afford to be *ad hoc* in design.

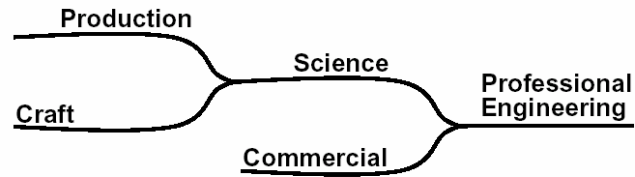
- How do you get predictability in software design?
- How do they get predictability in other disciplines?

It's tough to make predictions, especially about the future. - Yogi Berra

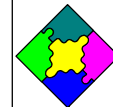
4



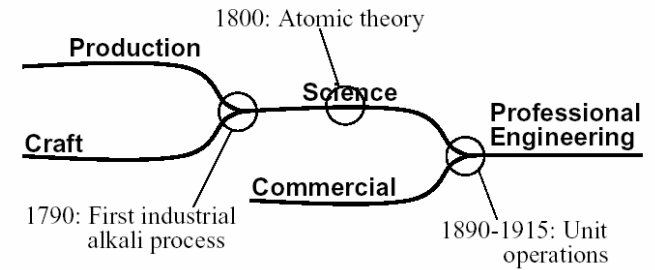
A Discipline of Engineering



5



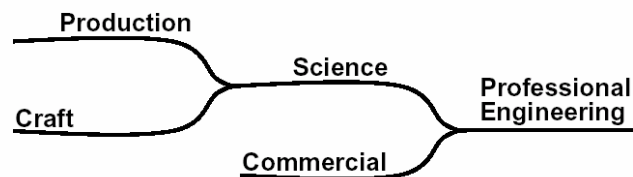
Chemical Engineering



6

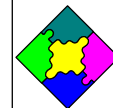


Software Engineering?



The disciplined application of scientific knowledge to resolve conflicting constraints and requirements for problems of immediate practical significance.

7



Does Software Engineering Proffer Predictability?

Are our systems built and run predictably?
Do we understand/predict/manage designs for their quality attributes?

Moral: software architects don't always understand how to engineer in quality attributes and to manage risk.

8



Engineering Design

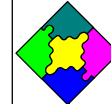
Routine. *Not* innovative.

Depends upon the existence of a mature discipline:

- | standard notation for recording and communicating designs
- | established procedures, published processes, design standards
- | small number of unit operations
- | widely disseminated handbooks

One man's "magic" is another man's engineering. - Robert Heinlein

9



Engineering Quality Attributes

We need to identify and analyze risks at the stage of architecture design.

To do this we need suitable architecture analysis techniques.

And we need analyzable designs.

How do we get these?

What about design patterns?

10



Design Patterns as a Foundation

The OO community has pioneered design patterns. Design patterns bring us closer to being an engineering discipline.

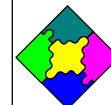
Q: What is missing?

A: Principled ways of analyzing them.

Design patterns give us design reuse.

We also need *analysis* reuse.

11



Rat-hole Avoidance

Rat-hole 1: Analysis \neq OO Analysis?

By "analysis" I mean a way of repeatedly determining the attribute-specific responses of a design to stimuli.

Rat-hole 2: Architecture \neq Design?

I believe that design is an activity.

Architecture, or architectural design is design at a high level of abstraction.

12



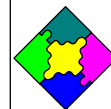
Architectural Styles

Software architecture gives us architectural styles (similar to OODP but often higher abstraction).

An architectural style consists of:

- component/connector types, topology
- constraints on the topology and behavior
- an *informal* description of the costs and benefits of the style, e.g.: "Use the pipe and filter style when reuse is desired and performance is not a top priority"

13



Architectural Styles as a Foundation

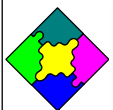
Architectural styles bring us closer to being an engineering discipline.

Q: What is missing?

A: Principled ways of analyzing them.

Once again, we have design reuse without any rigorous analysis reuse.

14



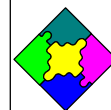
Building Upon Styles and Design Patterns: ABASs

Architectural styles and design patterns are a wonderful (and necessary) idea. They describe the essence of a recurring design problem and its solution.

Attribute-based architectural styles (ABASs) add *explicit quality attribute analysis models* to reason about the costs/benefits of a pattern or style.

No matter how complicated a problem is, it usually can be reduced to a simple, comprehensible form which is often the best solution. - An Wang

15



ABAS Motivation

Why add a quality attribute-based modeling framework to an architectural style?

- to make architectural design more routine and predictable
- to have a standard set of important attribute-based analysis questions associated with the style
- to tighten the link between design and analysis

16



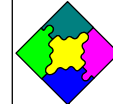
Analysis Models

To aid in structuring an ABAS and in understanding each attribute, we are using *attribute characterizations*.

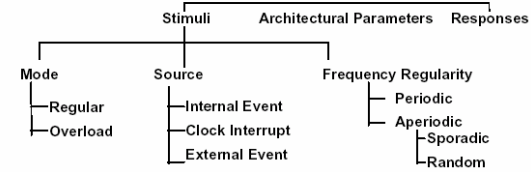


For each attribute, the characterization describes:

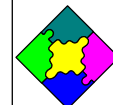
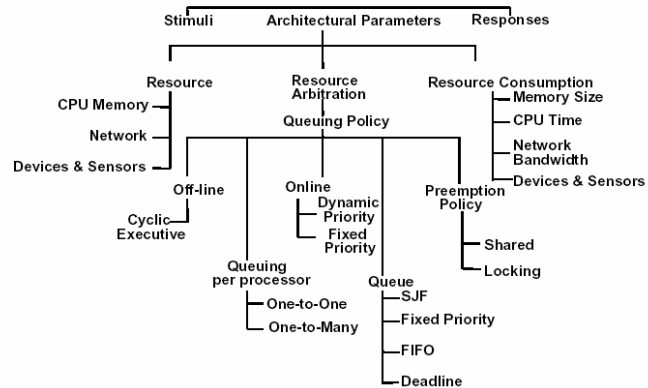
- the stimuli of interest
- the architectural style (and its parameters)
- the responses



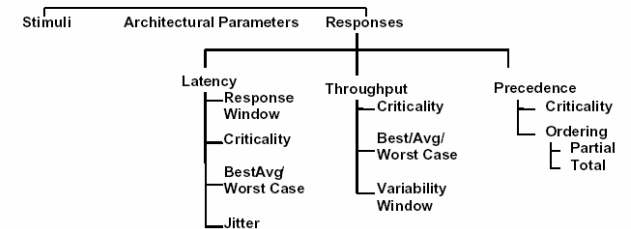
Performance Characterization



Performance Characterization



Performance Characterization

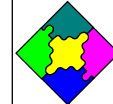




An ABAS Handbook

ABASs bridge the gap between program level and system level design.
To be useful, they must be documented.

To be good is noble; but to show others how to be good is nobler and less trouble. - Mark Twain



Example Handbook Contents

Performance:

- | Concurrent Pipelines
- | Multiple Messages
- | Synchronization
- | Cache
- | Client/Server

Modifiability:

- | Abstract Data Repository
- | Layers
- | Publish/Subscribe
- | Data Indirection

Availability:

- | Analytic Redundancy
- | Simplex
- | Trimodular Redundancy

Security:

- | Firewall
- | Virtual Private Network
- | Encryption/Decryption

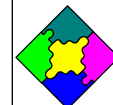
Usability:

- | Undo
- | Cancel
- | Visualization

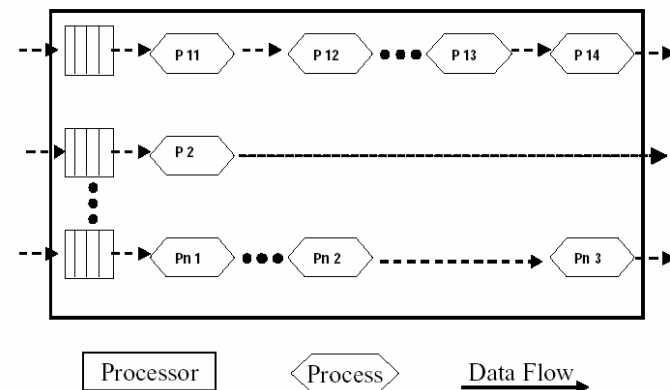


Parts of an ABAS

1. **Problem Description and Criteria:** characteristics of the problem solved.
2. **Stimuli/Responses:** the ABAS's quality attribute specific stimuli and the measures of the responses.
3. **Architectural Style:** components, connectors, parameters, topology, constraints.
4. **Analysis:** formally relating quality attribute models to the style; heuristics for reasoning about the style.



An Example ABAS - Concurrent Pipelines



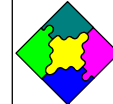


Concurrent Pipelines: Problem Description

Classical pipe and filter systems are used when a sequence of transformations is applied to a stream of data.

Concurrent pipelines are needed when multiple streams occur with real-time requirements. It is common for systems to use a sequence of processes to implement the sequence of transformations, sometimes with unintended performance consequences.

25



Concurrent Pipelines: Stimuli/Responses

Stimuli: one or more periodic or sporadic input streams

Responses: end-to-end worst case latency

A problem well stated is half solved. -
Charles Kettering

26



Concurrent Pipelines: Architectural Style

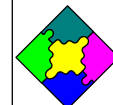
Components: filters implemented as independent processes

Connectors: pipes (data transmission)

Topology: pipeline

Constraints: pipes have bounded queues; no cycles in the topology; filters retain no state between invocations; filters incrementally transform input to output

27



Concurrent Pipelines: Analysis (Parameters)

Stimuli

- | rate of input arrivals ($1/T$)

Architectural parameters

- | queuing policy: e.g. FIFO
- | scheduling policy: e.g. on-line fixed priority
- | execution time (C) for each process
- | priority associated with each process

Responses

- | end-to-end deadlines

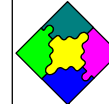
28



Concurrent Pipelines: Analysis (Guidelines)

- Determine the priority of the lowest priority process in the i^{th} pipeline, denoted by $LowP_i$.
- Determine the set of pipelines whose lowest priority process has a priority $> LowP_i$.
- Determine the set of pipelines that start with processes priorities $> LowP_i$ but eventually drop below $LowP_i$. Call this set HL.
- Determine the set of pipelines that start with processes priorities $< LowP_i$ but eventually rise above $LowP_i$. Call this set LH.

29



Concurrent Pipelines: Analysis (Formula)

Calculate the worst-case latency (L) for the i^{th} pipeline using the following formula as the basis for iteration.

$$L_{n+1} = \sum_{j \in H} \left\lceil \frac{L_n}{T_j} \right\rceil C_j + C_i + \sum_{j \in HL} C_j + \max_{j \in LH} (C_j)$$

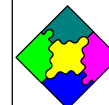
30



Concurrent Pipelines: Analysis (Heuristics)

- ▮ Latency is very sensitive to the $LowP_i$, the lowest priority process in the pipeline.
- ▮ How does your choice of priority assignment impact latency?
- ▮ Is there another prioritization strategy that might reduce latency (should the priority of the lowest priority process in each pipeline be changed)?
- ▮ Is the architecture flexible enough to accommodate later reprioritization?
- ▮ Is the effect of reallocating functionality to processes easily understood?

31



Using ABASs in Analysis

Analysis reuse:

- ▮ Provides standard questions to ask of each design, based upon the *responses* that are to be controlled.
- ▮ Propose models for evaluating and comparing designs; considering the alternatives and choosing the best one.

The wise man doesn't give the right answers, he poses the right questions.
- Claude Levi-Strauss

32



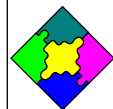
Using ABASs in Design

Design reuse:

- Building blocks for design (road tested).
- Packaged attribute-based reasoning about critical design decisions.
- Indexable by topology and response.

We must recognize the strong and undeniable influence that our language exerts on our ways of thinking and, in fact, delimits the abstract space in which we can formulate-- give form to--our thoughts. - Nicklas Wirth

33



ABASs are Analysis-centric

The analysis is typically both formal and informal.

We need both for practical use!

As far as the laws of mathematics refer to reality, they are not certain; as far as they are certain, they do not refer to reality. - Albert Einstein

34



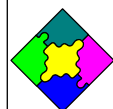
ABASs are Attribute-specific

We will have:

- client/server (performance)
- client/server (modifiability)

Each ABAS is characterized by its topology (the same) and its stimulus/response model (different).

35



ABASs are Abstract

ABASs:

- transcend any specific language or programming paradigm
- help us understand problems by "chunking"
- focuses attention on the aspects that are quality attribute specific
- help us find risks

Marking dynamos for repair: \$10,000.
2 hours labor: \$10; knowing where to mark: \$9,990. - Charles Steinmetz

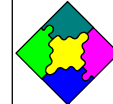
36



ABASs in Design and Analysis: An Example

This example will focus on the design side of the design/analysis coin.
An embedded real-time patient monitoring system.

37



Requirements

Portability:

- multiple hardware platforms
- use different types of devices (displays and sensors)
- run on different operating systems
- accommodate different user interface toolkits

38

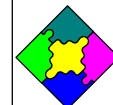


Requirements

Modifiability:

- adding/modifying producers and consumers of data: changes to smoothing/filtering algorithms, alarm sensing logic, and trend tracking
- new data formats
- different displays
- relocation of application entities to other processors

39



Requirements

Performance:

- Raw data is sensed every 10 ms. and must be displayed within 20 ms. of acquisition.
- Waveform calculations are performed every 100 ms./300 ms.
- Display results of 300 ms. calculations on the patient monitor as a waveform with an associated beep within 50 ms.
- Certain detectable conditions should cause an alarm to be triggered. The alarm should manifest on the patient monitor within 500

40



Applying ABASs

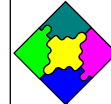
We will show how we build up the design from 3 ABASs.

- ▮ layering
- ▮ publish/subscribe
- ▮ concurrent pipelines

Q: How did we choose these ABASs?

A: Matching the quality attribute requirements with the stimuli/response models and topology in the ABASs

41



Layering: Criteria

Choose this ABAS if your problem inherently has distinguishable broad categories of functionality that:

- ▮ are internally highly coherent
- ▮ are stable with respect to changes
- ▮ depend upon each other in predictable ways
- ▮ do not have cycles of dependencies
- ▮ have low coupling with (at most two) other categories

42

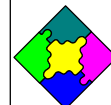


Layering: Stimuli/Responses

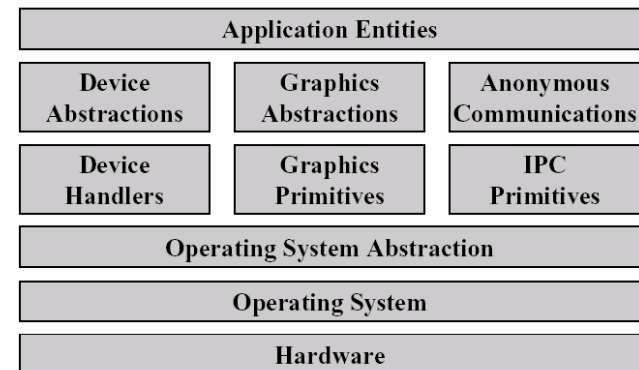
Stimulus: a change to a layer in the software

Responses: number of layers affected and number of components, interfaces, and connections added, deleted, and modified, along with a characterization of the complexity of these changes/deletions/modifications

43



Layering: Applying the ABAS



44

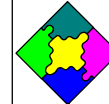


Publish/Subscribe: Criteria

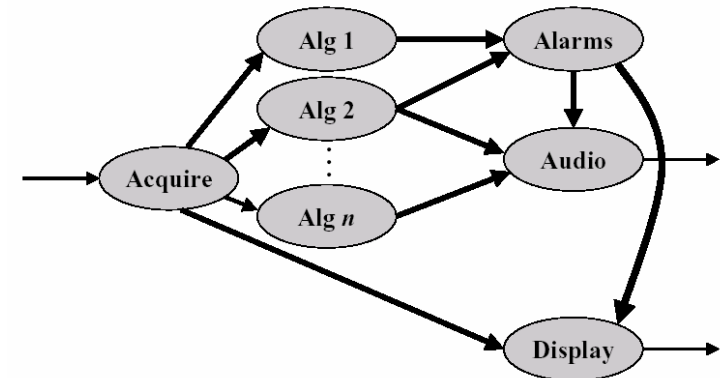
Choose this ABAS if the following conditions are true:

- ▮ Data producers will change the format of the data that they produce.
- ▮ The number and identity of producers and consumers of a data item is unknown or is likely to change.
- ▮ The ordering between producers/consumers is unknown or frequently changes.
- ▮ There are no complex temporal dependencies or tight real-time deadlines.

45



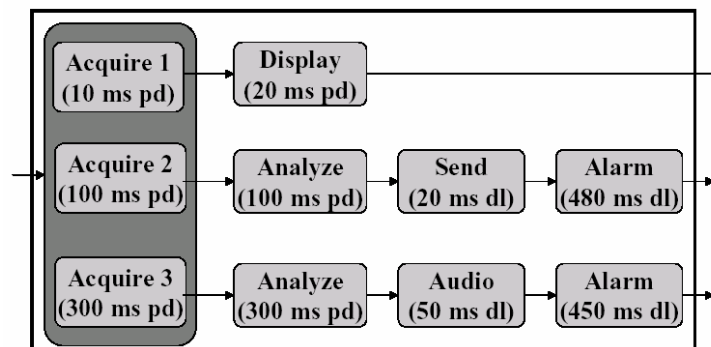
Publish/Subscribe: Applying the ABAS



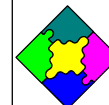
46



Concurrent Pipelines: Applying the ABAS



47



Concurrent Pipelines: Refining the Design

The next question is how to assign priorities to each process.

The ABAS provides us with several guidelines:

- ▮ the effective priority of a pipeline is the lowest priority in the pipeline
- ▮ shorter deadlines are generally accorded high priorities

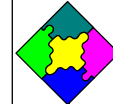
48



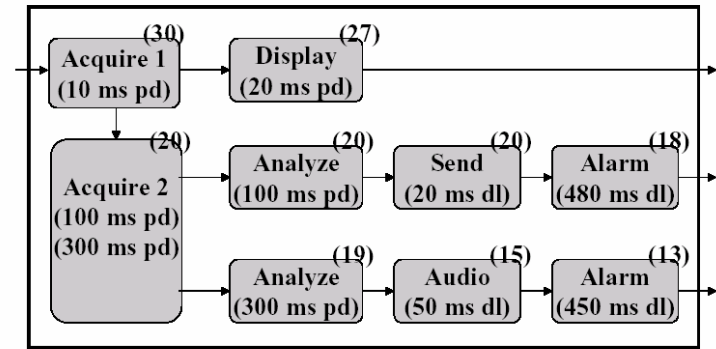
Concurrent Pipelines: Refining the Design

We can now begin to compute latency.
 Problem! Publish/Subscribe overhead is
 2-3 ms. per call => can't always meet
 Acquire's 10 ms. period.
 Solution: split Acquire into two processes.

49



Concurrent Pipelines: Refining the Design



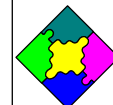
50



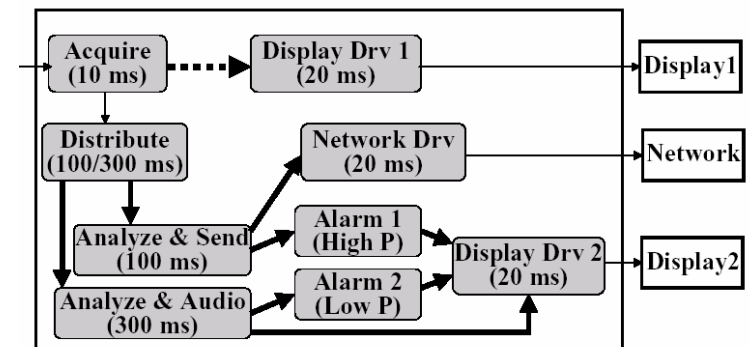
Combining ABASs & Analyses

Problem! Publish/Subscribe is still too
 costly for the 10 ms. Acquire process.
 Solution: use IPC directly.
 But this means layer bridging and a
 performance/modifiability tradeoff.

51



The Final Design (Excerpt): Concurrency/Hardware View



52



Skeptic's Corner: Is it any good?

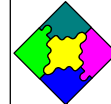
Efficiency:

- We created the initial architecture in 1-2 hours (for a 1M LOC system).
- We fleshed out the architecture in 1 week.

Risk:

- We found all of the major problems:
 - needing to split Acquire into 2 processes
 - layer bridging.

53



So What?

Isn't this just good engineering?

YES!!!!

Is it regularly practiced?

NO!

Where there is doubt, there is freedom.
- Latin proverb

54



ABASs Guide Design

Design must start from somewhere.

Currently we give architects little but their own imaginations and experience to start with.

ABASs provide concrete designs and analyses: a start for *reasoning* about architecture and quality attributes.

Example isn't another way to teach, it is the only way to teach - Albert Einstein

55