Architecture Rationale

Charles L. Chen & Danhua Shao

March 21, 2006

Overview

➔Motivation

➔The CBSP Approach

➔Archium

➔Using CBSP and Archium

➔Conclusions

➔Questions

Motivation

➔How do you go from requirements to architecture?

➔How do you capture the rationale for the architectural decisions that are made in this process?

The CBSP Approach

"Reconciling Software Requirements and Architectures With Intermediate Models"

by Paul Grunbacher, Alexander Egyed, and Nenad Medvidovic

The CBSP Approach

➔Helps architects bridge the gap between requirements and architecture

➔Evaluate the relevance of a requirement along the 6 CBSP dimensions

➔Refine requirements into architecturally friendly CBSP artifacts

The 6 CBSP Dimensions

1. C – Components
2. B – Bus (Connector)
3. S – System
4. CP – Component Property
5. BP – Bus Property
6. SP – System Property

Relevance of Requirements

➔Determine a set of core requirements through stakeholder-based prioritization

➔Architects evaluate the relevance of the core requirements by ranking them 0 (irrelevant) to 3 (fully relevant) along the 6 CBSP dimensions

➔Use Kendall's coefficient of concordance to determine consensus among architects and discuss any conflicts

Requirements to CBSP Artifacts

➔Split requirements

↳R: The system should provide an interface to a Web browser.

↳C: A Web browser should be used as a component in the system.

↳B: A connector should be provided to ensure interoperability with 3rd party components.

Requirements to CBSP Artifacts

➔Combine requirements

- R1: Support for different types of Cargo
- R2: Support for cargo arrival and vehicle estimation
- C_d: Data component to represent Cargo

Requirements to CBSP Artifacts

➔Make requirements more specific

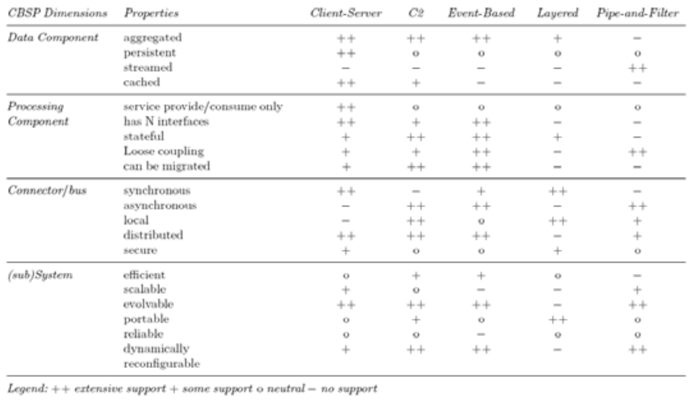
- R: Updates to system functionality should be enabled with minimal downtime.
- BP: Robust connectors should be provided to facilitate runtime component addition and removal.

➔Generalize requirements

- R: Spreadsheet data must be encrypted when dispatched across the network.
- SP: The system should be secure.

(or perhaps - S: The system should transmit data securely.)

Choosing an Architectural Style



CBSP Dimensions	*Properties*	*Client-Server*	*C2*	*Event-Based*	*Layered*	*Pipe-and-Filter*
Data Component	aggregated	++	++	++	+	−
	persistent	++	o	o	o	o
	streamed	−	−	−	−	++
	cached	++	+	−	−	−
Processing Component	service provide/consume only	++	o	o	o	o
	has N interfaces	++	+	++	−	−
	stateful	+	++	++	+	−
	Loose coupling	+	+	++	−	++
	can be migrated	+	++	++	−	−
Connector/bus	synchronous	++	−	+	++	−
	asynchronous	−	++	++	−	++
	local	−	++	o	++	+
	distributed	++	++	++	−	+
	secure	+	o	o	+	o
(sub)System	efficient	o	+	+	o	−
	scalable	+	o	−	−	+
	evolvable	++	++	++	−	++
	portable	o	+	o	++	o
	reliable	o	o	−	o	o
	dynamically reconfigurable	+	++	++	−	++

Legend: ++ extensive support + some support o neutral − no support

Tool Support for CBSP

➔Selection of requirements

- Distributed voting tool
- Requirements rated on relevance and feasibility
- Automatic classification as "low hanging fruit," "important with hurdles," "maybe later," and "forget them"

➔Architectural classification of requirements

- Fully tool supported with a COTS voting tool

Tool Support for CBSP

➔Identifying and resolving conflicts

- Automatic highlighting of conflicts
- Graph showing the vote spread

➔Architectural refinement

- Translate CBSP into a UML representation
- Traceability between requirements and CBSP artifacts

➔Trade-off choices

- None (yet)

Cargo Router Case Study

➔Route cargo from delivery ports to warehouses

➔Provide reports & estimates of cargo arrival times & vehicle status

Cargo Router Case Study

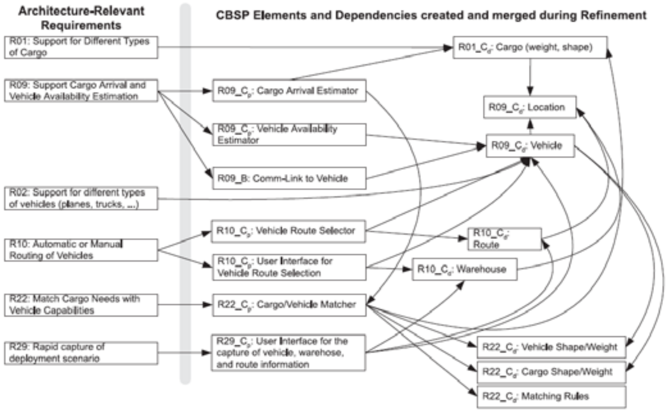
➔Selection of requirements

- Initially: 81 requirements
- After review and merging: 64 requirements
- After joint prioritization: 25 requirements

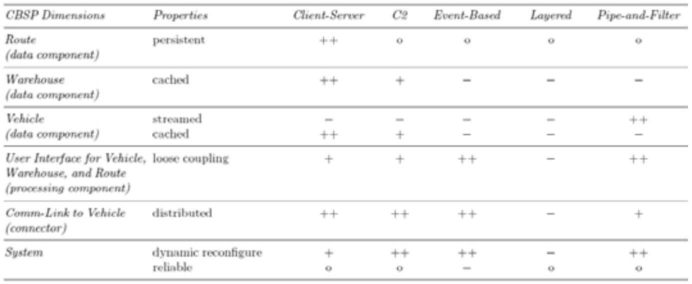
➔Architectural classification of requirements

- Conflict -> Discussion -> Clarification

Cargo Router Case Study

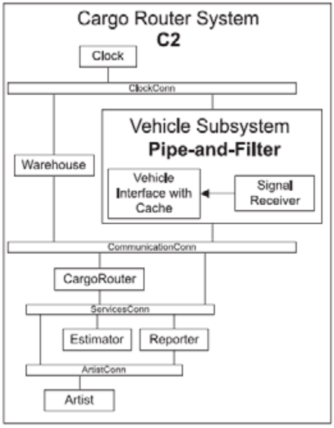

Architecture-Relevant Requirements
CBSP Elements and Dependencies created and merged during Refinement
R01: Support for Different Types of Cargo
R09: Support Cargo Arrival and Vehicle Availability Estimation
R02: Support for different types of vehicles (planes, trucks, ...)
R10: Automatic or Manual Routing of Vehicles
R22: Match Cargo Needs with Vehicle Capabilities
R29: Rapid capture of deployment scenario
R09_C_p: Cargo Arrival Estimator
R09_C_p: Vehicle Availability Estimator
R09_B: Comm-Link to Vehicle
R10_C_p: Vehicle Route Selector
R10_C_p: User Interface for Vehicle Route Selection
R22_C_p: Cargo/Vehicle Matcher
R29_C_p: User Interface for the capture of vehicle, warehose, and route information
R01_C_d: Cargo (weight, shape)
R09_C_d: Location
R09_C_d: Vehicle
R10_C_d: Route
R10_C_d: Warehouse
R22_C_d: Vehicle Shape/Weight
R22_C_d: Cargo Shape/Weight
R22_C_d: Matching Rules

Cargo Router Case Study



CBSP Dimensions	*Properties*	*Client-Server*	*C2*	*Event-Based*	*Layered*	*Pipe-and-Filter*
Route *(data component)*	persistent	++	o	o	o	o
Warehouse *(data component)*	cached	++	+	–	–	–
Vehicle *(data component)*	streamed cached	– ++	– +	– –	– –	++ –
User Interface for Vehicle, Warehouse, and Route (processing component)	loose coupling	+	+	++	–	++
Comm-Link to Vehicle *(connector)*	distributed	++	++	++	–	+
System	dynamic reconfigure reliable	+ o	++ o	++ –	– o	++ o

Cargo Router Case Study



Summary

➔CBSP helps architects bridge the gap between requirements and architecture

➔ Relevance of requirements to architecture along the CBSP dimensions

Archium

"Software Architecture as a Set of Architectural Design Decisions"

by Anton Jansen and Jan Bosch

Outline

➔ Introduction

➔ Architectural Design Decisions

➔ Archium

➔ Case Study: Athena

➔ Summary

Introduction

➔Common view of software architecture:

Component + Connector

➔Problem: How to react to changes?

➔Reason: Decision knowledge vaporization.

Proposed Solution

➔New view of software architecture:

A composition of a set of architectural design decisions

➔ Questions: What is a design decision? How to document it?

Architectural Design Decisions (1)

➔Architectural design decision includes:

↳Rationale: why the change is made

↳Design rules: what should be followed

↳Design constraints: what should NOT be allowed

↳Additional requirements: new requirements resulting from the change

Architectural Design Decisions (2)

➔Architectural design decisions can help:

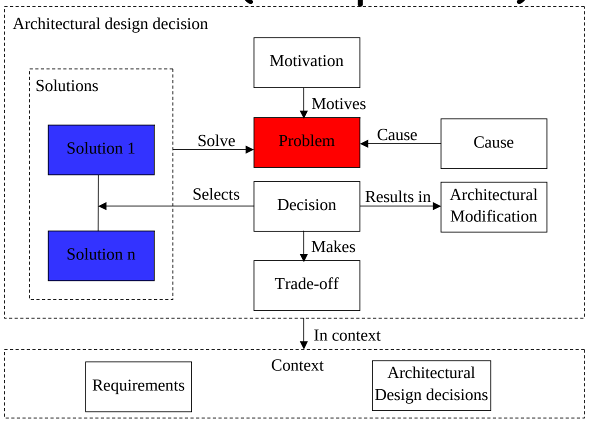
- Find and make changes
- Check for the violation of design rules and constraints
- Remove obsolete design decisions

Architectural Design Decisions (3)

➔Decisions documents should have:

- First class architectural design decisions
- Explicit architectural changes
- Support for modification, subtraction, and addition
- Clear relationship between architecture and realization
- First class architectural concept

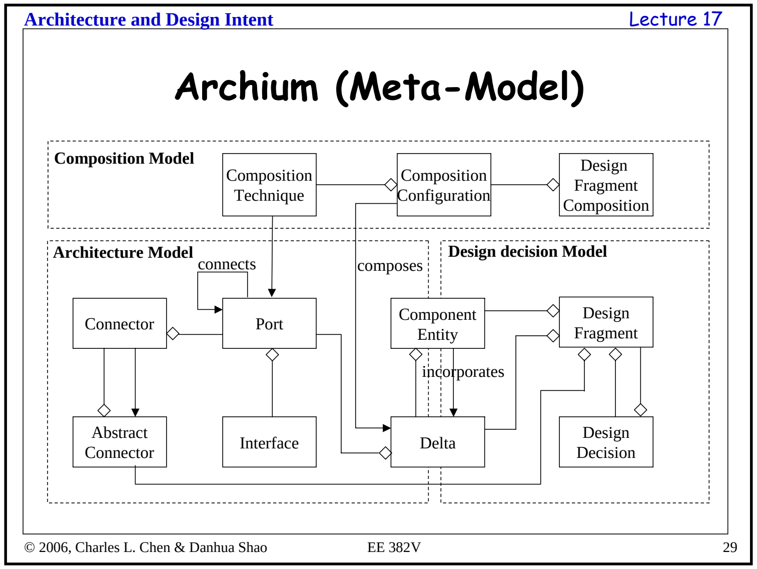
Archium (Concept Model)



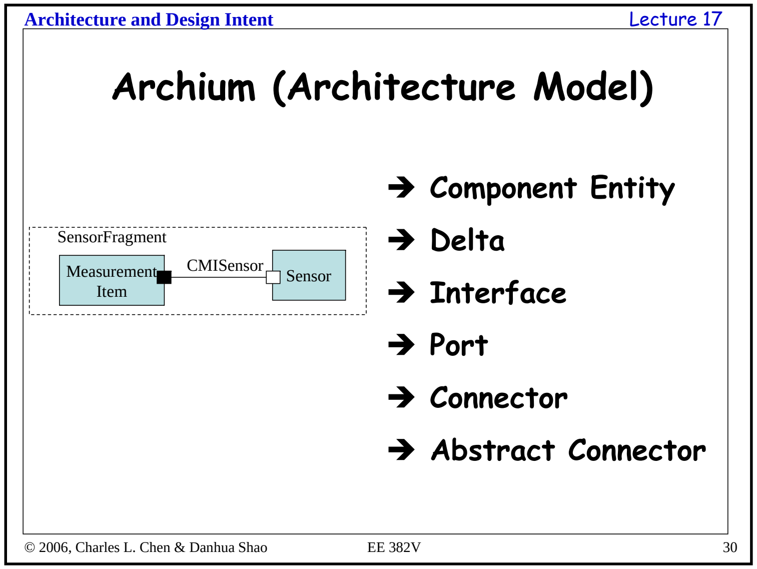
Archium (Solution)

➔Solution includes:

- Description
- Design rules
- Design constraints
- Consequences
- Pros
- Cons

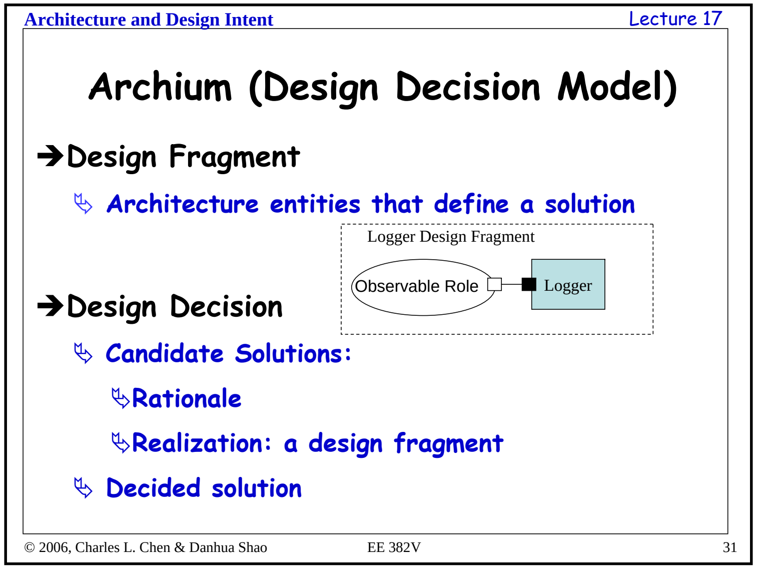


Archium (Meta-Model)



Archium (Architecture Model)

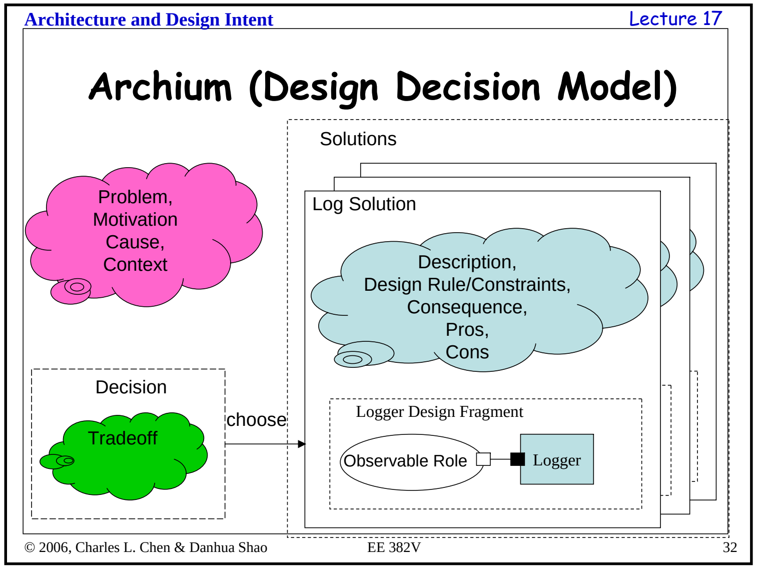
- ➔ Component Entity
- ➔ Delta
- ➔ Interface
- ➔ Port
- ➔ Connector
- ➔ Abstract Connector



Archium (Design Decision Model)

- ➔ Design Fragment
  - ↳ Architecture entities that define a solution

- ➔ Design Decision
  - ↳ Candidate Solutions:
    - ↳ Rationale
    - ↳ Realization: a design fragment
  - ↳ Decided solution



Archium (Design Decision Model)

Solutions
Problem, Motivation Cause, Context
Log Solution
Description, Design Rule/Constraints, Consequence, Pros, Cons
Decision
Tradeoff
choose
Logger Design Fragment
Observable Role
Logger

Archium (Composition Model)

➔ Apply the design decision to architecture elements

➔ Three parts:

- ↳ Composition Technique: change on port
- ↳ Composition Configuration: change on entity
- ↳ Design Fragment Composition: change on design fragment

Archium (Composition Model)

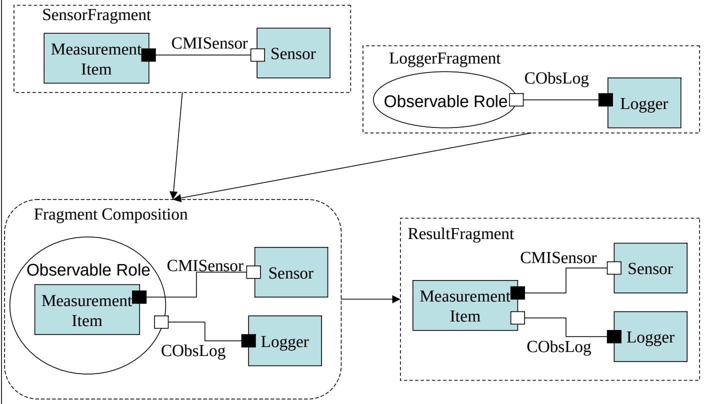
➔ Composition:

- ↳ context design fragment + design decision = new design fragment

➔ Eg.

SensorFragment + LoggerFragment = LoggedSensorFragment

Archium (Composition Model)



A Case Study: Athena

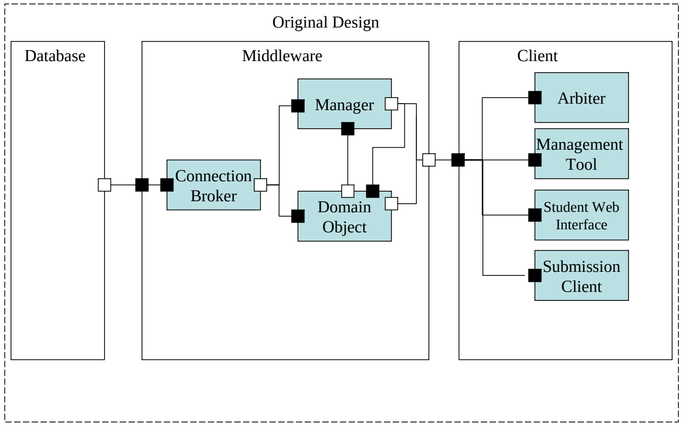
➔ A submission system

judge, review, manipulate, and archive programs

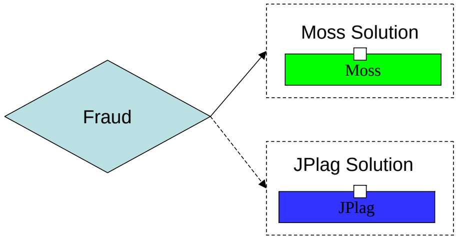
➔ Three-tiered architecture

Database, middleware, client

Fraud Design Decision (1)



Fraud Design Decision (2)



Fraud Design Decision (3)

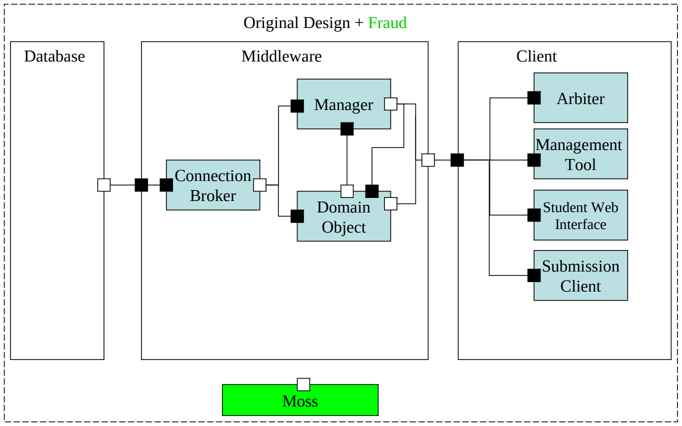
➔Moss:

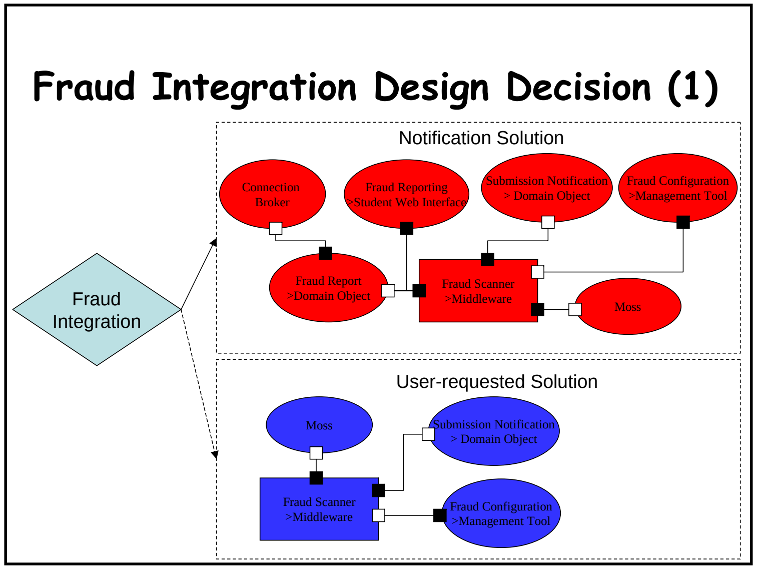
- ↳Description
  - ↳C/S
- ↳Design rules
  - ↳assignment clear
- ↳Design constraints
  - ↳Batch Model
- ↳Consequences
  - ↳Add Moss Server
- ↳Pros
  - ↳Strong in fraud detection
  - ↳Multiple prog lang
  - ↳Free
- ↳Cons
  - ↳Add integration part

➔JPlag:

- ↳Description
  - ↳C/S
- ↳Design rules
  - ↳assignment clear
- ↳Design constraints
  - ↳Batch Model
- ↳Consequences
  - ↳Add JPlag Server
- ↳Pros
  - ↳Free
- ↳Cons
  - ↳Small prog lang
  - ↳Add integration part
  - ↳No demo

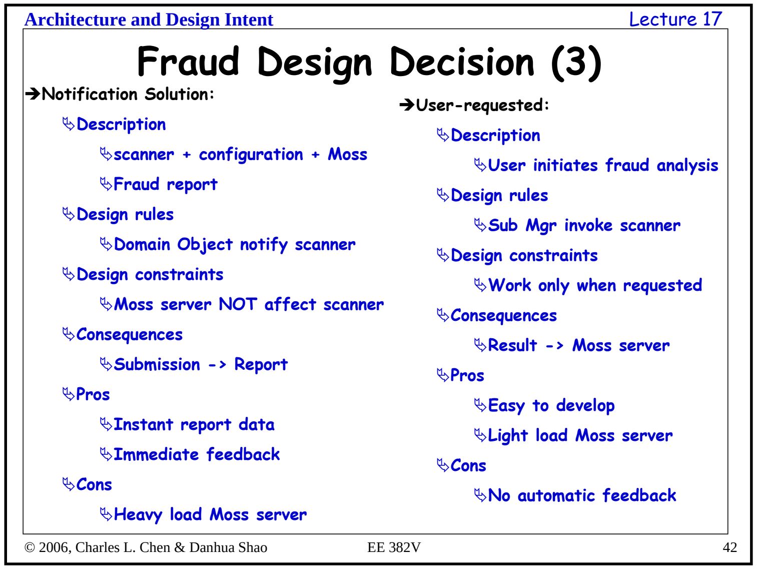
Fraud Design Decision (4)





Fraud Integration Design Decision (1)

Fraud Integration
Notification Solution
Connection Broker
Fraud Reporting >Student Web Interface
Submission Notification > Domain Object
Fraud Configuration >Management Tool
Fraud Report >Domain Object
Fraud Scanner >Middleware
Moss
User-requested Solution
Moss
Submission Notification > Domain Object
Fraud Scanner >Middleware
Fraud Configuration >Management Tool



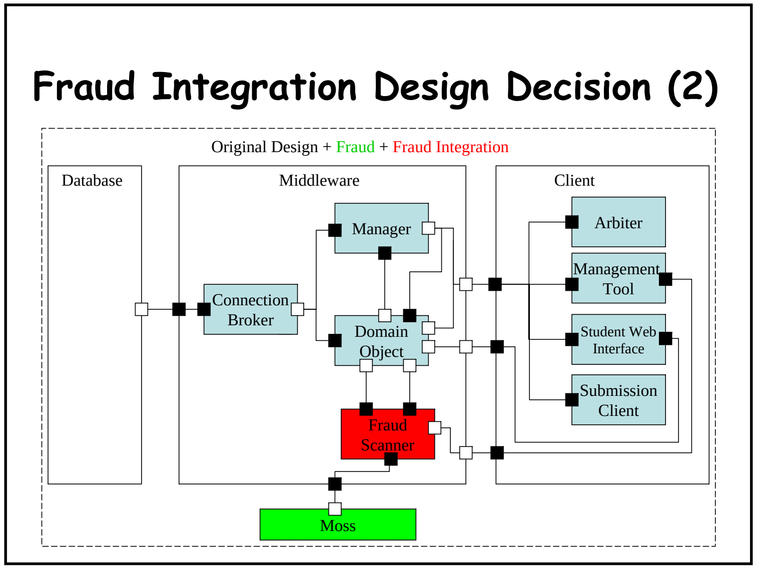
Fraud Design Decision (3)

➔Notification Solution:

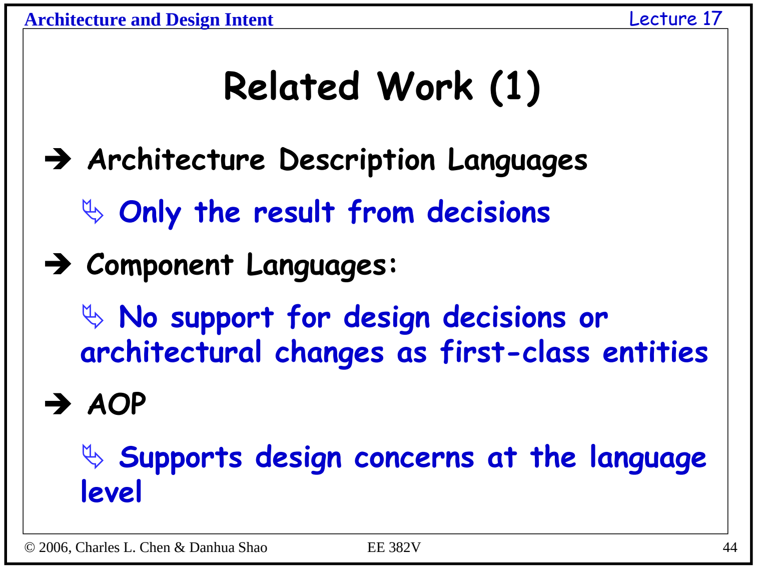
- ↳Description
  - ↳scanner + configuration + Moss
  - ↳Fraud report
- ↳Design rules
  - ↳Domain Object notify scanner
- ↳Design constraints
  - ↳Moss server NOT affect scanner
- ↳Consequences
  - ↳Submission -> Report
- ↳Pros
  - ↳Instant report data
  - ↳Immediate feedback
- ↳Cons
  - ↳Heavy load Moss server

➔User-requested:

- ↳Description
  - ↳User initiates fraud analysis
- ↳Design rules
  - ↳Sub Mgr invoke scanner
- ↳Design constraints
  - ↳Work only when requested
- ↳Consequences
  - ↳Result -> Moss server
- ↳Pros
  - ↳Easy to develop
  - ↳Light load Moss server
- ↳Cons
  - ↳No automatic feedback



Fraud Integration Design Decision (2)



Related Work (1)

➔ Architecture Description Languages

- ↳ Only the result from decisions

➔ Component Languages:

- ↳ No support for design decisions or architectural changes as first-class entities

➔ AOP

- ↳ Supports design concerns at the language level

Related Work (2)

- ➔ Design Pattern
  - ↳ Realization part in Archium
- ➔ Knowledge System
  - ↳ Not integrated with architectural model

Summary

- ➔A new view of software architecture: evolution with a set of design decisions
- ➔ Archium model: document architecture with notations of deltas, design fragments, and design decisions
- ➔ Visualize the whole process

Reference

- ➔"Evaluation of Tool Support for Architectural Evolution", Anton Jansen and Jan Bosch.
- ➔ http://www.archium.net/
- ➔ http://wiki.zefhemel.com/index.php/Archium

Using CBSP and Archium

- ➔Background
  - ↳Evolving the CLC-4-TTS Suite
  - ↳Adding speech property support for FreeTTS
  - ↳First attempt fell short – problem with race conditions
  - ↳Very general idea of how to approach the problem; no specific designs for a solution

Using CBSP

- Requirements
  - R1: Speech properties must be configurable for FreeTTS.
  - R2: Users must be able to interact with the system at all times.
  - R3: Speech may not have any long pauses.
  - R4: Equal priority messages should be spoken in the order they were received.
  - R5: High priority messages must be able preempt lower priority messages; preempted messages do not have to be saved.
  - R6: Speech properties from one message may not interfere with those from another.

Using CBSP

- Refinement into CBSP artifacts
  - R1 - R1_C: SetProperties interface to FreeTTS
  - R2 - SP: Suggests multithreading
  - R3 - Eliminated: Definition of "long pause" + queuing system
  - R4 - R4_B: Queue for messages
  - R5 - R5_C: Queue Manager
  - R6 - BP: Suggests using a queue that can handle messages and associated properties
  - C2 Style (?)

Using Archium

- Problem
  - The current interface to FreeTTS does not allow speech properties to be set
- Motivation
  - The result of this is that Linux and Mac users are unable to experience the CSS speech property support being introduced
- Cause
  - Speech property support has not been implemented yet
- Context
  - Evolving the existing CLC-4-TTS Suite

Using Archium

- Potential solution #1: JSML Generator
  - Description: Use JSML to encode the properties into a string along with the message. Pass the entire thing into FreeTTS.
  - Design rules: All generated strings must be well formed JSML strings.
  - Design constraints: Message needs to be put within tags that contain the properties; therefore messages and associated properties should be delivered at the same time.
  - Consequences: CLC-4-TTS Suite is dependent on FreeTTS supporting JSML.

Using Archium

- Potential solution #1 (cont.)
  - Pros:
    - +Easy to code (similar system exists for SAPI 5 already)
    - +FreeTTS manages the queue
    - +Easy to force FreeTTS to empty queue (for prioritization)
  - Cons:
    - -FreeTTS does not yet support JSML; significant wait time expected as the FreeTTS project appears to be in hiatus (last update was in February 2005).

Using Archium

- Potential Solution #2: Queue System
  - Description: Create a queue system that will set the speech properties for FreeTTS, pass FreeTTS a message to be spoken, and then wait until it is ready for a new message with a different set of speech properties.
  - Design rules: Must keep track of messages and associated speech properties
  - Design constraints: Queue must not interfere with users' ability to interact with the system as whole; blocking is only to block the speech portion but nothing else.

Using Archium

- Potential solution #2 (cont.)
  - Consequences: CLC-4-TTS Suite is dependent on Java FreeTTS allowing the setting of speech properties.
  - Pros:
    - +Can be implemented immediately as Java FreeTTS already allows for the setting of speech properties.
  - Cons:
    - -Far more difficult than using a JSML generator

CBSP

- Stengths
  - Structured process of going from requirements to CBSP artifacts
  - CBSP artifacts can be traced back to requirements

- Weaknesses
  - Evaluations for trade-off choices focus on choosing an architectural style after having derived the CBSP artifacts, but not on deriving the CBSP artifacts themselves

CBSP

➔Comments

↳Rejecting the JSML Generator – had the idea, unsure where to put the rationale for rejecting it in the CBSP approach

Archium

➔Stengths

↳Alternate solutions and the reasons for choosing one solution over another are explicitly captured

↳Pros and cons of a solution are documented as part of the solution

➔Weaknesses

↳No real help given on thinking up the potential solutions

Archium

➔Comments

↳Had the benefit of knowing the problem – not sure how easy Archium would have been to use otherwise since there is no guidance in arriving at potential solutions given the Problem, Motivation, Cause, and Context

Results

➔Solutions

↳Similar solutions with both methods

↳May be an artifact of both methods being used by the same person

➔Strengths and Weaknesses

↳CBSP and Archium each had their respective strengths and weaknesses

➔Little / No Tool Support (?)

↳Did not find any tool support for CBSP

↳Tool support for Archium did not work

Conclusion

➔ Rationale is an approach to address the change and evolution problems in software architectural design

➔ Rationale can be documented as an intermediate model to refine requirements and architecture designs

Conclusion

➔Challenges for rationale

- ➥ Check the validity of rationale
- ➥ Capture rationale from the requirements and architectural design
- ➥ Integrate with architecture description languages and tools

Questions?