Architectural Design Drivers

Presented by:

Sahil Thaker

Shounak Roychowdhury

Outline

- Motivation
- Paper: Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Designs
- Paper: From System Goals to Software Architecture
- Quality Drivers
- Paper: Understanding Architectural Influences and Decisions in large System Projects
- Flight Simulator Case Study Example

Motivation

- Architecture Design has an impact on NFR
  - Security, fault tolerance, performance, maintainability, interoperability, etc.
- How do we map Functional and Non Functional Requirements to characteristics of Architecture?

Motivation for This Research

- What we have:
  - ADLs
    - Components, Connectors, Rules for Interactions
  - Rationale Documentation
  - Verification
- Goal similar to:
  - On the Criteria To Be Used in Decomposing Systems into Modules - D.L. Parnas
- Heuristic to guide design of Architectures
- Understand rationale behind architectural decisions
- Predictability

Design Decision: Data Structures

- Analysis based on established algorithmic theory
- Requirements
  - Operations
  - Optimize for performance, space
  - Distribution of operations
- Analysis
  - Space/time complexity
  - Amortized analysis

Using NFR to Select Among Alternatives in Architectural Designs

Using NFR to Select Among Alternatives in Architectural Designs

- NFR-Framework
  - NFRs are represented as goals
  - "Methods" are used to organize NFR-related knowledge
    - Decomposition
    - Satisficing
    - Argumentation
  - Uses correlation rules to evaluate architectural alternatives
  - Evaluate effects of each design decision

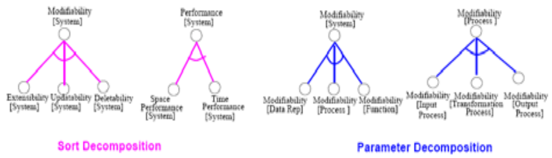
Goals

- Very modifiable system
- Good system performance

- *Modifiability[system; critical]*
  - Type*(of goal)* [parameter list; importance]

Methods

- Decomposition methods



- Satisficing methods
  - Use correlation rules and architectural patterns to satisfy goals
- Argumentation methods
  - Codify Rationale
  - Technique not mentioned

Correlation Rules

	Shared Data	Abstract Data Type	Implicit Invocation	Pipe & Filter
Modifiability [Process]	− −	−	+	+
Modifiability [Data Rep]	− −	+	−	− −
Space Performance	++		−	− −*
Time Performance		−	− −	
Reusability	−	+	+−	+

− −*: if size of data in actual domain is huge

Table 1: Correlation Table, based on Garlan and Shaw.

Goal Graph

Figure 3: Initial stage of goal graph for KWIC architectural design (based on Garlan and Shaw).

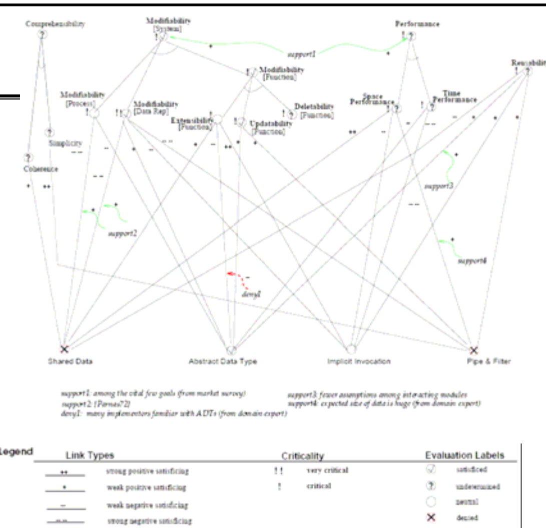


Figure 5: Goal graph selecting among architectural alternatives for a KWIC system.

From System Goals to Software Architecture

From System Goals to Software Architecture

- Requirements elicitation
  - Derive goals to be achieved by the system
  - WHY issues
- Operationalization of goals into specifications
  - WHAT issues
- Assignment of responsibilities for spec to agents (human, devices, software)
  - WHO issues
- Architectural Design
  - Structural Issues

Goal-Oriented Architectural Derivation

- Requirements on the GO Process
  - Systematic (traceable)
  - Incremental, allow reasoning on partial models
  - At least “arguable” or at best “provably” correct and good architectures
  - Allow different views to be highlighted
    - Security, fault tolerance view, etc.

Steps

1. Derive Goal Graph through Refinement
2. Goal -> Requirements
3. Requirements -> Specs
4. Specs -> Abstract Dataflow Architecture
5. Style-based Architectural Refinement
6. Pattern-based Architectural Refinement

Background: Terminology

- Goal: Prescriptive statement of intent
- Agent: active components
  - Human, device, software components, etc.
  - Software vs. Environment
- Domain Properties: Descriptive statements about Environment
  - Physical laws, organizational norms
- Functional Goals: Services to be provided
- Non Functional Goals
  - QOS: safety, security, usability, performance
  - Development goals: maintainability, reusability, etc.
  - Architectural Constraints: constraints on environment
    - distribution of human agents, physical devices

Background..

- Requirement: A goal under responsibility of an agent in the software
- Expectation: A goal under responsibility of an agent in the environment
- Softgoals: prescribe preferred behavior

From System Goals to Software Requirements

- Derivation process
  - Goal modeling
    - Goal refinement graph
    - Refined into AND/OR structured sub-goals
  - Object modeling
    - e.g. UML
  - Agent modeling
    - Identify and attach to goals
  - Operationalization
    - Identify operations, pre/post conditions and trigger conditions (obligations)

Portion of Goal Refinement Graph for a Meeting Schedule System

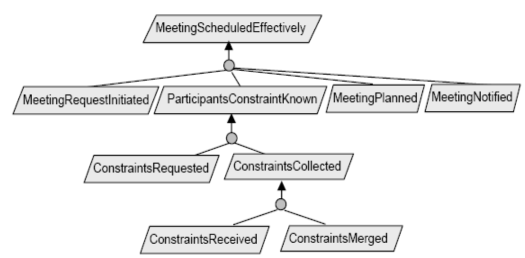


Fig. 1 – Portion of a goal refinement graph

Refinement and Operationalization

Goal *ParticipantsConstraintsKnown* was refined using Refine-by-Milestone pattern

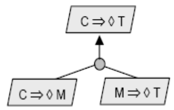


Fig. 2 – Refinement-by-milestone pattern

Goal *ConstraintsRequested* was operationalized into an operation *RequestConstraintsToParticipants* using Bounded-achieve pattern.

The operation specification prescribes that ¬T becomes T as soon as C ∧ ¬T holds for d−1 time units

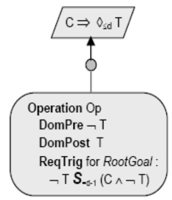


Fig. 3 – Bounded-Achieve operationalization pattern

Intertwining Between Requirements and Architecture

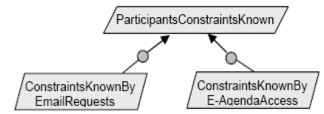


Fig. 4 – Alternative goal refinements

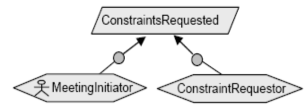


Fig. 5 – Alternative agent assignments

From Requirements to Specification

Requirement Achieve [ConstraintsRequested]

FormalSpec ∀m: Meeting, p: Participant:

Requested (m) ∧ Invited (p, m) ⇒ $\Diamond_{\leq Rd}$ ConstrRequested (p)

In this formulation, the associations Requested, Invited and ConstrRequested correspond to phenomena that are observable in the environment. They need to be mapped to software input-ouput variables to produce, e.g., the following target software specification:

∀m: MeetingClass, p: ParticipantClass

MeetRequest (m) ∧ p **in** InviteeList (m) ⇒ $\Diamond_{\leq Rd}$ ConstrReqSent (p)

For our above example, the accuracy goals will be

∀m: Meeting, m': MeetingClass, p: Participant, p': ParticipantClass

Mapping (m, m') ∧ Mapping (p, p') ⇒

MeetRequest (m) ⇔ Requested (m)

p' **in** InviteeList (m') ⇔ Invited (p, m)

ConstrReqSent (p) ⇔ ConstrRequested (p)

From Specs to Abstract Dataflow Architectures

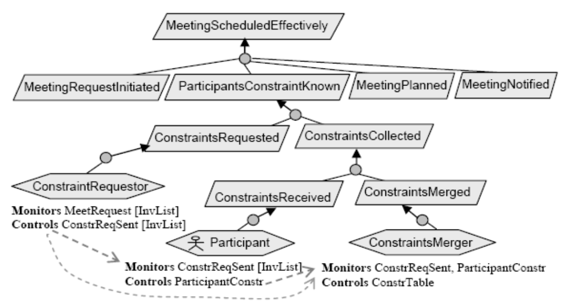


Fig. 6 – Assigned agents, their interfaces and data dependencies

Derived Dataflow Architecture

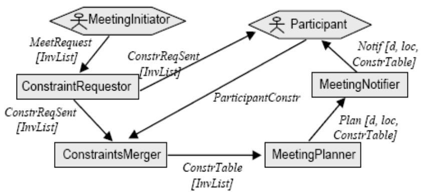


Fig. 7 – Derived dataflow architecture

Style-based Architectural Refinement to meet Architectural Constraints

- Refine dataflow architecture by imposing suitable architectural styles
  - Styles whose underlying softgoals match architectural constraints
- Refinements must preserve the properties of more abstract connectors and components

Event-based Architectural Style

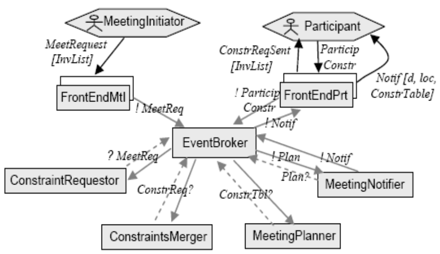
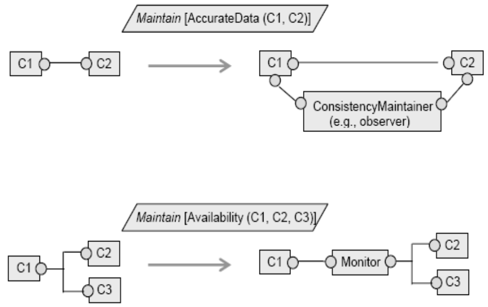


Fig. 9 – Style-based architecture to meet architectural constraints

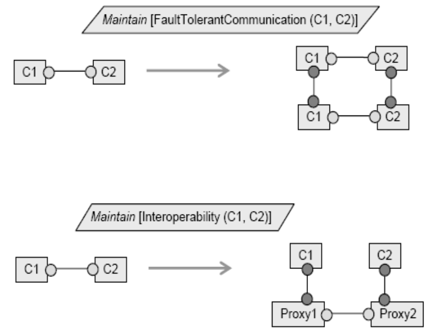
Pattern-based Refinement to Achieve Non-Functional Requirements

- EventBroker should be split into several brokers handling different kinds of events if *Maximize[Cohesion(EventBroker)]* is to be achieved
- Security goals restrict information flows along (secure) channels
- Accuracy goals impose interactions to maintain consistent state between objects

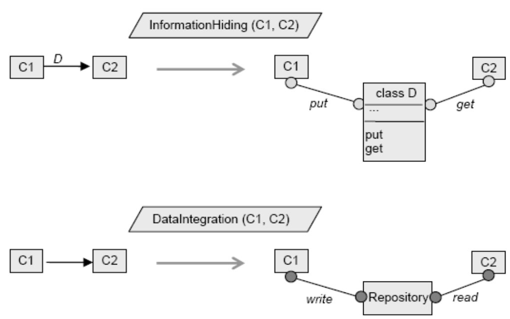
Architectural refinement patterns for quality-of-service goals



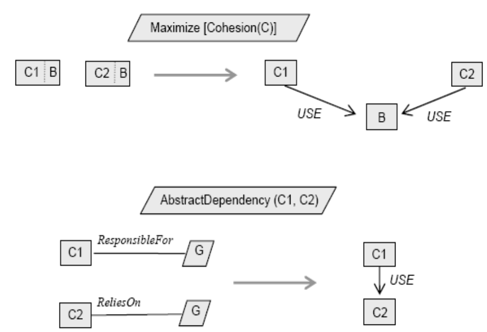
Architectural refinement patterns for quality-of-service goals



Architectural refinement patterns for development goals



Architectural refinement patterns for development goals



Outline for Second Half

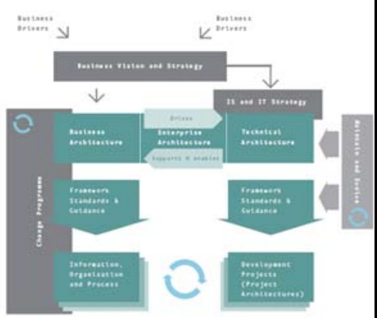
- Study of Architectural Drivers (influences)
  - System requirements
  - Quality
  - Goals
  - Designers' experience
  - Organization's culture
- Paper Discussion
  - Paul Clements, "Understanding Architectural Influences and Decisions in large System Projects," *In Proceedings of ICSE 17*, Workshop on Software Architecture, 1995
- Case Study
  - Integrability as an architectural driver for flight simulator design

Quality Attributes in Architecture

- Achievement of quality attributes is critical for the success of any system
- Architecture by itself cannot achieve qualities
- Qualities act as guide for architectural design
- Types of qualities
  - Business attributes
  - Architectural attributes
  - System attributes

Business Quality Attributes

- Business issues
  - Competitive pressures
  - Functionality differentiators
  - Targeted releases
- Cost/Benefit
  - Technology
  - Expertise
    - » In-house
    - » outsource
- Scalability
  - Users
  - Graceful degradation



Architectural Quality Attributes

- Availability
  - System's available time
- Usability
  - Usage criterion
- Modifiability
  - Modification criterion
- Performance
  - Runtime measure
- Security
  - Prevention of unauthorized usage
- Testability
  - Testing criterion
- Integrability
  - Seamless integration of large systems

Integrability

- Arises as a driving concern in large systems
  - In Database system designs
  - In Large Enterprise Resource Planning (ERP) applications
- Loose coupling or minimal dependencies between elements
  - Easier to coordinate, evaluate, independent testing
- Especially those developed by distributed teams or separate organizations
  - Across countries, continents
- Using componentization
  - Assimilation of components and deliverables

System Quality Attributes

- Measures system's characteristics
- Enables system designers to make reasonable assumptions for better system prediction
- Failure to address can lead to dire consequences
- Allow to develop systematic way to relate system architecture's objective decision & design trade-offs

Paper Synopsis

- Architectural influences in large projects
  - Architecture as summary of architectural decisions
    - » Rationale for component selection, interconnection mechanism, architectural styles, real-time, etc.
- Hypothesis
  - Architecture as function of influencing factors
  - Set of influences is at least partially enumerable
  - The architecture is the summary result of a set of component decisions made by an architect
  - Set of decisions is at least partially enumerable
  - Possible correlation between drivers and architectural decisions

Study of Large System Architectures

- Study of engineering practices of successful architectures
- Examples
  - Initial Sector Suite System (ISSS)
    - 10^6 lines of code for air traffic control to process radar and flight plan data in real time
  - CelsiusTech
    - Shipboard fire control system (common architecture & reusable components)
  - Prism
    - Generic architecture for US military
  - GenVoca
    - Product-line high performance database systems
  - Structural Modeling at SEI
    - Common patterns in various application domains (flight simulator)

Architectural Influences (1/3)

- Project-related Influences
  - Time-independent functional requirements
    - » Measurable in some form, verified against some standards
  - Performance requirements
  - Functional quality requirements } (quality attributes)
  - Afunctional requirements } (quality attributes)
    - » that cannot be measured – openness, maintainability, portability
  - Driving requirements (difficult to satisfy)
- Axioms:
  - P1: The driving *afunctional* requirements are a major influence in the architecture chosen
  - P2: The driving *functional quality* requirements are a major influence in the architecture chosen
  - P3: The driving performance requirements are a major influence in the architecture chosen
  - P4: Driving functional requirements, other than those relating to functional quality attributes, are not usually a major influence in a system's architecture

Driving Requirements

Table 1: The driving requirements of the case-study set.

Case study	Primary requirement	Secondary requirements
ISSS (FAA)	Ultra-high availability	Performance, safety, usability
RCS (NIST)	Safety	Performance
CelsiusTech	Product line development	Performance
GenVoca	Short time to market	Performance
PRISM	Reuse	Information security, performance
Structural modeling	Scalability, integrability	Performance

Architectural Influences (2/3)

- Organization-related influences
  - Goals & background of developing organization
  - Organization policies
- Axioms
  - O1: The existence of tools and/or capital infrastructure tailored to particular architectures will exert a bias towards those architectures
    - » .NET or Java shop?
  - O2: Organizational goals, such as mandate to reuse existing products or a desire to evolve the developing systems into a product line, will exert a major influence on the chosen architecture
  - O3: Organization's development history, as evidenced by architectures of systems developed previously by that organization, will exert a secondary influence on the chosen architecture.

Architectural Influences (3/3)

- Architecture related influences
  - If an architect has solved the problem in a particular approach, it is likely to be used again
- Axiom
  - A1: Architectures previously used by the project's architecture will exert a major influence on the chosen architecture. The influence (positive or negative) will be directly proportional to the perceived success or failure of the prior efforts

Cataloging Influences

Table 2: Problem viewpoints from Shaw [10]

Functional decomposition	Data flow	Object-oriented
State machine	Event-oriented	Process control
Decision table	Data structure	

Table 3: Blum's taxonomy of software development methods [2]

	Problem-oriented	Product-oriented
Conceptual	I Structured analysis Entity-relationship model Logical construction of systems Modern structured analysis Object-oriented analysis	II Structured design Object-oriented design
Formal	III PSL/PSA JSD[a] VDM[b]	IV Levels of abstraction Step-wise refinement Proof of correctness Data abstraction JSP[c] Object-oriented programming

a. JSD: Jackson System Development Method
b. VDM: Vienna Development Method
c. JSP: Jackson System Programming

Table 4: Extracting influences to test hypotheses

Axiom	Pertains to	Criteria for data extraction
P1 P2	Afunctional requirements Functional quality requirements	ISO Standard 9126 [4]; Driving or nondriving
P3	Performance requirements	Source: communication, computation; Driving or nondriving
P4	Functional requirements	Shaw on shared-information system architectures [11]
O1	Tools and infrastructure	Presence of tools that support a particular architectural style
O2	Organizational goals	Presence of reuse repository; desire to engineer product line
O3	Organizational history	Shaw [10] and Blum [2]
A1	Architect's history	Shaw [10] and Blum [2]

Correlation Influences & Decisions

- Axioms
  - – C1: Static architectural decisions tend to affect afunctional properties; hence, driving afunctional requirements tend to motivate the static architectural decisions. There will be an observable correlation between driving afunctional requirements and static architectural decisions
  - – C2: Dynamic architectural decisions tend to affect performance properties; hence, driving performance requirements tend to motivate the dynamic architectural decisions. There will be an observable correlation between driving performance requirements and dynamic architectural decisions

Flight Simulator*

- Integrability as an architectural driver
- Modern flight Simulators
  - – are complex software system
  - With stringent functional concerns
    - – Real time performance
    - – Must be amenable to frequent update
  - With hard quality concerns
    - – Modifiability
      - » accommodate changes in requirements in simulated aircrafts and their environments
    - – Scalability of function
      - » Able to extend the system to simulate more of real-world
  - Careful attention is given to the software architecture in a complex domain to enable the construction of this system
    - – could be understood by a variety of software engineers
    - – were easy to integrate
    - – Amenable to downstream modifications

* L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice," CMU SEI Series, Pearson Education Inc., 2003

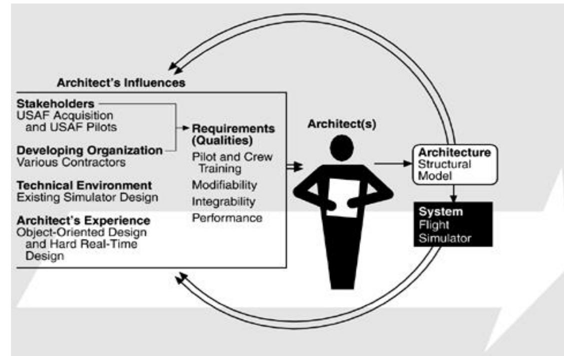
Flight Simulators

- Structural Model for Integrability
- Relationship to the Architecture Business Cycle
- Requirements and Qualities
- Architectural Solution

Structural Model for Integrability

- Structural model
  - Simplicity and similarity
- Decoupling of data- and control-passing strategies from computation
  - Allows easy integration among components
  - Adds scalability
- Minimizing number of modules
- A small number of system-wide coordination strategies
- Transparency of designs
- Other quality attributes are necessary for flight simulation

Relationship to Business Process



Requirements and Qualities (1/3)

- Role of the crew being trained
  - The purpose of a flight simulator is to instruct the pilot and crew
    - how to operate a particular aircraft
    - how to perform maneuvers such as mid-air refueling
    - how to respond to situations such as an attack on the aircraft
- Role of the environment
  - Typically the environment is a computer model
    - with multi-aircraft training exercises it can include individuals other than the pilot and crew
    - Other models like during simulating refueling, the (simulated) refueling aircraft introduces turbulence into the (modeled) atmosphere
- Role of simulation instructor
  - The instructor is responsible for monitoring the pilot's performance
    - initiating training situations.
    - Typical situations like malfunctions of equipment, attacks on the aircraft from foes, and weather conditions
    - Use a separate console to monitor the activities of the crew, to inject malfunctions into the aircraft, and to control the environment.

Requirements & Qualities (2/3)

- Models
  - The models used in the aircraft and the environment are capable of being simulated to almost arbitrary fidelity.
    - **Consequence**: desire to want more fidelity makes *performance* become one of the important quality requirements for a flight simulator
- States of Execution
  - » (A flight simulator can execute in several states.)
    - *Operate* corresponds to the normal functioning of the simulator as a training tool.
    - *Configure* is used when modifications must be made to a current training session. For example,from a single-aircraft exercise to mid-air refueling
    - *Halt* is used to stop the current simulation.
    - *Replay* uses a journal to move through the simulation without crew interaction."Record/playback"tactic used here

Requirements & Qualities (3/3)

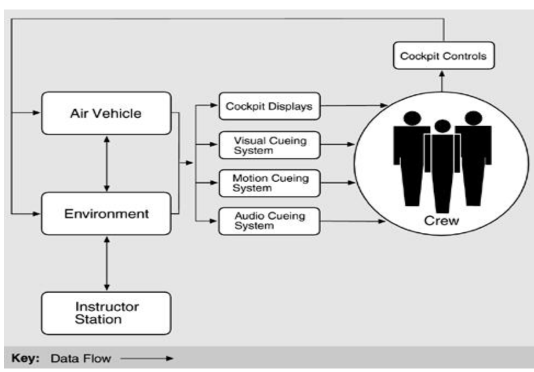
- **Real-time performance constraints**
  - Flight simulators must execute at fixed frame rates that are high enough to ensure fidelity.
- **Continuous development and modification**
  - To provide a realistic training experience, a flight simulator must be faithful to the actual air vehicles, which are continually being modified and updated.
- **Large size and high complexity**
  - The size can be millions of lines of code the complexity shows exponential growth trend
- **Developed in geographically distributed areas**
  - In either case, the Integrability is made more difficult because the paths of communication are long.

(Original v/s New) Strategies

- **"Original Strategy"**
  - Model based on task
  - 2 problems caused the U.S. Air Force to investigate new simulator designs
  - Very expensive debugging, testing, and modification.
    - **Consequence**: Integrability and modifiability emerged as a driving architectural concern.
  - Unclear mapping between software structure and aircraft structure.
    - **Consequence:** cause problems with both modifiability and integration.

- **"New Strategy"**
  - The architectural pattern, *Structural Modeling*, is an object-oriented design
  - Results from the reconsideration of the problems of earlier flight simulators
  - Models the subsystems and controller children of the air vehicle.
  - Add real-time scheduling to control the execution order of the simulation's subsystems to guaranteed fidelity.

Architectural Solution



Time Management

- Periodic time management to maintain real-time
  - A periodic time-management scheme has a fixed (simulated) time quantum based on the frame rate, that is the basis of scheduling the system processes
  - A simulation based on it will be able to keep simulated time and real time in synchronization
  - managed by adjusting the responsibilities of the individual processes small enough to be computed in the allocated quantum
- Event-based time management is used where real-time performance is not critical
  - such as the instructor station
  - An event-based time-management scheme similar to the interrupt-based scheduling used in many operating systems.
  - In this case, simulated time advances by the invoked processes placing events on the event queue and the scheduler choosing the next event to process.

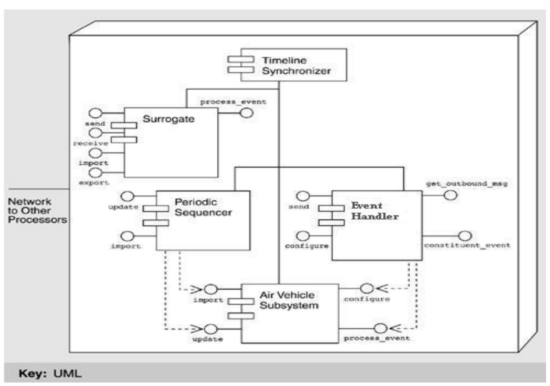
Scheduling

- The scheduling of the three models (portions) of the flight simulator
  - The instructor station model is typically scheduled on an event basis. Those events come from the instructor's interactions
  - The air vehicle model is scheduled on a periodic basis
  - The environment model can be scheduled using either way. A simple policy for managing events within a periodically scheduled processor is that -- after a synchronization step, periodic processing occur first and complete before any a-periodic processing
- Communication from the portions of the system managed on an event basis to the portions managed using periodic scheduling appears as a-periodic
  - Communication from the instructor station model to the air vehicle model appears as a-periodic

Architectural Pattern

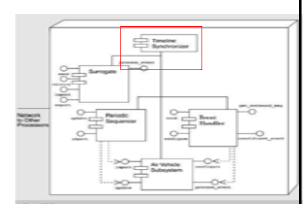
- Structural Model
  - Coarsest level
  - Developed at CMU's SEI
- Executive
  - Handles coordination & Synchronization
  - Real-time scheduling
- Application
  - Modeling the air vehicle

Model Executive



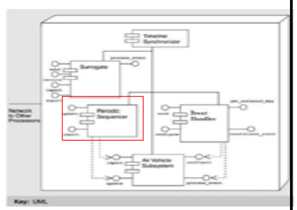
Timeline Synchronizer

- Base scheduling mechanism
- Maintains the simulation's internal notion of time
- Maintains the current state of the simulation
- Implements a scheduling policy for coordinating both periodic and a-periodic processing
- Coordinates time with other portions of the simulator



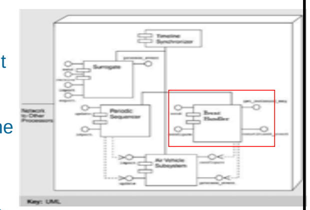
Periodic Sequencer

- Used to conduct all <u>periodic</u> processing performed by the simulator's subsystems
- This involves invoking the subsystems to perform periodic operations according to fixed schedules.
- Two operations to the timeline synchronizer
  - The *import* operation : invoke subsystems' import operation.
  - The *update* operation : invoke subsystems' update operations



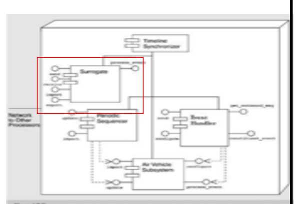
Event Handler

- Used to conduct all *a-periodic* processing performed by the simulation's subsystems.
- The event handler provides four operations to the timeline synchronizer:
  - <u>configure</u> : start a new training mission
  - <u>constituent_event</u> : used when an event is targeted for a particular instance of a module
  - <u>get_outbound_msg</u> : used by the timeline synchronizer to conduct a-periodic processing while in system operating states
  - <u>Send</u> : used by subsystem controllers to send events to other subsystem controllers and messages to other systems



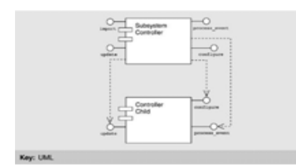
Surrogate

- Is an application that uses "use an intermediary" tactic
- Are responsible for system-to-system communication between the air vehicle model and the environment model or the instructor station model.
- Surrogates are aware of the physical details of the system with which they communicate
- Responsible for representation, communication protocol, and so forth



Air Vehicle Application

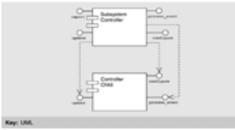
- Subsystem Controller
- Controller Child
- Data:
  - Subsystem controllers pass data to and from other subsystem controller instances and to their children.
  - Controller children pass data only to and from their parents, not to any other controller children.



- Control:
  - Controller children receive control only from their parents and return it only to their parents.

Subsystem Controller

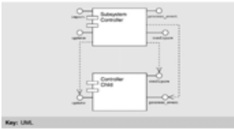
– Used to <u>interconnect a set of functionally related children</u> to do the following:
  – Achieve the simulation of a subsystem as a whole
  – Mediate control and a-periodic communication between the system & subsystems
  – Initialize themselves & their children
  – Route requests for malfunctions and the setting of simulation parameters to their children
  – Subsystem controllers *may* support the reconfiguration of mission parameters
  – Subsystem controllers realize these capabilities through *periodic* and *a-periodic* operations made available to the *periodic sequencer* and *event handler*, respectively



Controller Child

– In general, controller child support the simulation of an individual part, or object, within some functional assembly.

– Each child provides a simulation algorithm that determines its own state based on the following:
  – Its former state
  – Inputs that represent its connections with logically adjacent children
  – Some elapsed time interval
  – A child makes this determination when it is requested by its *subsystem controller*. This capability is called *updating*



Skeletal System

- The structural frame work above is the basis for a skeletal system for a flight simulator.
  – Jet fighter
  – Commercial aeroplane
  – Helicopter
- This is a general simulation framework that can be used for other simulator.
  – Nuclear reactor
- Modeling the flight simulator, a complex system by only six module types
  – makes the architecture (comparatively) simple to build, understand, integrate, grow, and modify.
- None of the details about functionality in it.
  – The process of making an actual simulation will be dictated by the functional partitioning process.

Allocating Functionality to Controller Children (1/4)

- How operational functionality is allocated to instances of the modules in that pattern.
- A functional partitioning process by defining instances of the subsystem controllers.
- This sample partitioning based on the underlying physical aircraft.

Allocating Functionality to Controller Children (2/3)

- Use an object-oriented decomposition approach
  - It maintains a close correspondence between the aircraft partitions and the simulator
  - provides us with a set of conceptual models that map closely to the real world.
  - A change in the aircraft is easily identifiable with aircraft functional partitions.

Allocating Functionality to Controller Children (3/4)

- The number and size of the simulator interfaces are reduced.
  - This derives from a strong semantic cohesion within partitions, placing the largest interfaces within partitions instead of across them
- Localization of malfunctions easy
  - they are associated with specific pieces of aircraft equipment.
  - It is easier to analyze the effects of malfunctions when dealing with this physical mapping.

Allocating Functionality to Controller Children (4/4)

- The airframe becomes the focal point
- Groups exist for the airframe can be specified by
  - Kinetics: elements that deal with forces exerted on the airframe
  - Aircraft systems: parts within the airframe provide the aircraft with power
  - Avionics: things that provide some ancillary support to the aircraft within the airframe
  - Environment: things associated with the environment in which the air vehicle model operates

Group Decomposition (1/2)

- The coarsest decomposition of the air vehicle model is the group
  - Groups decompose into systems, which in turn decompose into subsystems
  - Subsystems provide the instances of the subsystem controllers
- Groups and systems are not directly reflected in the architecture.
  - They are useful to organize the functionality assigned to the various instances of subsystem controllers.

Group Decomposition (2/2)

– n-Square Charts
  – One method of presenting information about the interfaces in a system
  – Easy to illustrate how the partitions relate to each other with this method.
  – A good method for capturing the input and output of a module and can illustrate the abstractions used in various parts of the design

Kinetics Group	Loads	Vehicle State Vector	Vehicle Position
Power	**Aircraft Systems Group**	Power	
Inertial State	Loads	**Avionics Group**	Ownship Emissions
Atmosphere, Terrain, and Weather Data		Environment Emitter Data	**Environment Group**

Realizing Goals

Table 8.1. How the Structural Modeling Pattern Achieves Its Goals

Goal	How Achieved	Tactics Used
Performance	Periodic scheduling strategy using time budgets	Static scheduling
Integrability	Separation of computation from coordination	Restrict communication
	Indirect data and control connections	Use intermediary
Modifiability	Few module types	Restrict communication
	Physically based decomposition	Semantic coherence
		Interface stability

Conclusions

- Qualities as architectural drivers
- Discussion regarding the paper
- Integrability as an architectural driver: a case study