Architectural design

- Software design (1980s(?)) – Start with a concept, and in a top down manner, break the concept into smaller and smaller chucks until each chunk is both precise and its implementation details well understood.
- Current view – For each system, determine the underlying model (i.e., Its architecture), and then use that to guide design and development.

Why use architectural designs?

- Reason about system behavior (function, performance, reliability, …)
- Provide infrastructure for making future decisions (what can be changed without losing system integrity, …)

Architectural descriptions contain:

- System structure – high level computational elements and their interactions
- Interaction abstractions – ability to discuss interactions among components of the design (e.g., pipe, client-server, event broadcasting, database protocols)
- Global properties – Describe overall system behavior (end-to-end data rates, system reliability, performance), not component behavior

Architectural styles

Dataflow systems
- Batch sequential
- Pipes and filters

Call and return systems
- Main program and subroutine
- Object—oriented system
- Layered system

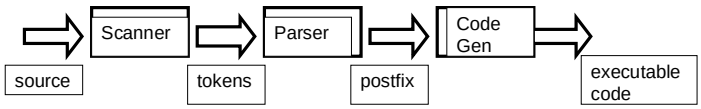
Independent components
- Communicating processes
- Client server systems
- Event systems

Virtual machines
- Interpreter
- Rule-based systems
- Process control systems

Data-centered repositories
- Databases
- Blackboard system
- Hypertext systems

Goals for architectural designs

Each style provides:
- Vocabulary of design elements
- Design rules or constraints
- Semantic interpretations (e.g., what is a “pipelined decomposition of a compiler?”)



Why have these?

- Promotes design reuse
- Code reuse (e.g., use common infrastructure pieces, UNIX pipes, X-Windows)
- Easier to understand
- Supports interoperability (e.g., CORBA, OSI)
- Allows for verification of higher order features (e.g., protocols for pipelined systems)

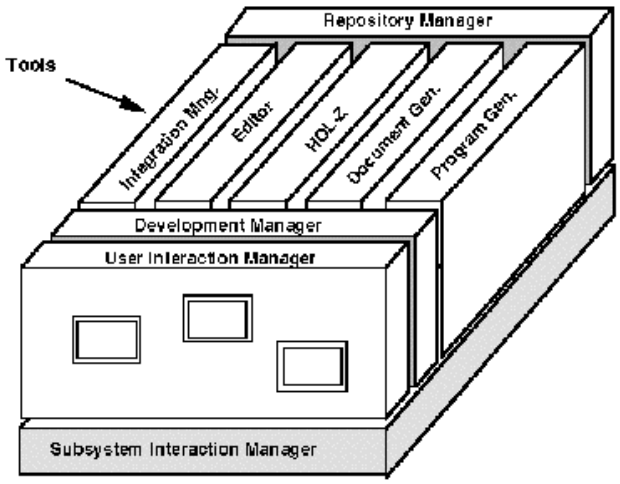
NIST/ECMA Framework Reference Model

A framework provides a common set of services needed to support application programs written for the domain of the environment. For a software engineering environment that means framework services support tools which aid in the design, development, deployment and management of the software development process.

In 1988 ECMA began work on describing such a framework reference model. The initial version was published in 1990. It included heavy emphasis on the data repository aspects of the framework.

Beginning in 1989, NIST's Integrated Software Engineering Environment (ISEE) Working Group, working jointly with ECMA, extended the model to include a greater variety of services. The original model was revised in 1991 and again in 1993.

The model is just a catalog of services that may be applicable to an environment framework. There was no implied architecture in describing any of the services. HOWEVER, it is often used as a preliminary architecture for such systems.

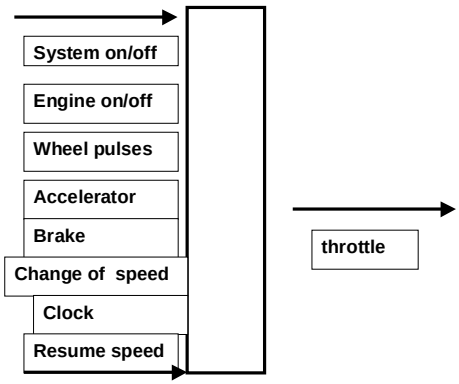


Reference Model Services

1. The 21 object management services provide for the storage, retrieval and management of a persistent object store.
2. The six process management services provide for developing process models of the development life cycle and the management of these processes within the environment.
3. The five communication services provide for communication among tools using communication paths like messages, RPCs and shared data storage.
4. The eight policy enforcement services provide the confidentiality, security and integrity requirements on an environment.
5. The ten user interface services provide the interface between the executing program and the user who is interacting with the environment.
6. The six framework administration services provide for the installation and tailoring of tools, users and other objects into the framework.
7. The ten operating system services provide basic system functionality.

Example: Cruise Control

IEEE Software (Shaw, 11/95)



Cruise control - II

System on/off – control cruise control

Engine on/off – car active

Wheel pulses – one per revolution

Brake – if pressed, cruise control suspended

Accelerator – How far accelerator depressed

Change of speed – Increase or decrease speed if cruise control is on

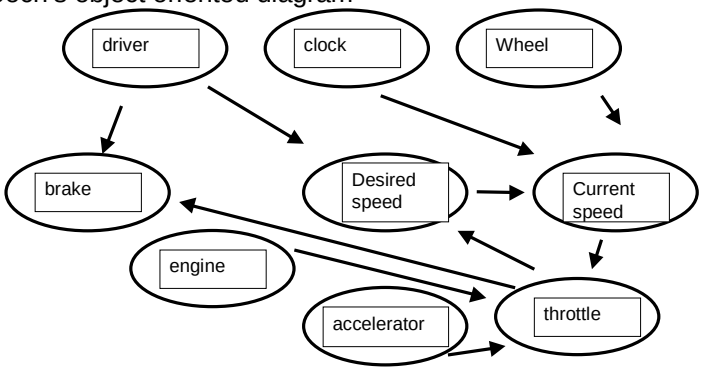
Resume speed – If cruise control is on, last maintained speed resumes

Clock – Pulse every millisecond

Example Architectural Styles

Object—oriented architectures: Systems are discrete objects that encapsulate state and definitions of operations.

Similar to the encapsulated data types previously discussed. The various models of OO design differ in how they define objects, define sequencing among objects, and how a design satisfies requirements. – Booch's object oriented diagram



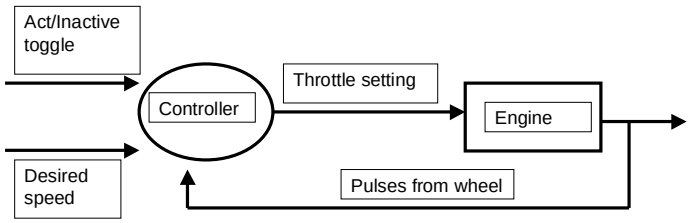
State-Based architecture

Focus on the major modes, or states, of the system, and on the events that cause transition between 2 states -- example: Example Gannon – Atlee

Current mode		Ignited	Running	Too fast	Brake	Act.	Deact Resume	New mode
Off	@T	---	---	---	---	---	---	Inactive
Inactive	@F	---	---	---	---	---	---	Off
	t	t	---	f	@T	---	---	Cruise
Cruise	@F	---	---	---	---	---	---	Off
	---	@F	---	---	---	---	---	Inactive
	---	---	@T	---	---	---	---	
	---	---	---	@T	---	---	---	Override
	---	---	---	---	---	@T	---	
Override	@F	---	---	---	---	---	---	Off
	---	@F	---	---	---	---	---	Inactive
	t	t	---	F	@T	---	---	Cruise
	t	t	---	f	---	---	@T	

Feedback control architectures

A special form of dataflow architecture adapted for embedded systems. They control a process by continuously available signals. (Shaw design)



REAL-TIME ARCHITECTURES

Add stringent response time characteristics. Each column is set of processes to execute for that state. (Ward and Keskar design)

Process/Action	Capture current speed	Increase speed	Maintain speed
Stop accelerating	0	0	0
Accelerate	0	1	0
Maintain speed	0	0	1
Accel. To desired speed	0	1	0
Get and maintain speed	1	0	1
Stop maintaining speed	0	0	0
Turn CC off	0	0	0
Cruise control ready	0	0	0

DESIGN EVALUATION CRITERIA

(Question: How do these relate to the PBR questions in an inspection?)

Separation of concerns:

- Does design separate independent parts of the system?
- Is redundant information avoided?
- Are closely related parts grouped together? Locality allows for an "interchangeable parts" strategy.

Generally:

- OO focuses on real-world entities
- State designs focus on a system's operational modes
- Process-control designs focus on feedback relationships
- Real-time designs focus on events and their order of execution

DESIGN EVALUATION CRITERIA-II

Perspicuity of design:

- Does the expression of the design correspond to the problem being solved?
- Is there traceability to the important requirements?

Ability to check and analyze design:

- Is the design easy to analyze and check for correctness, performance, and other properties of interest?
- If more than one model is used, is the interaction between them easy to follow?

Abstraction power:

- Does the design highlight and hide appropriate details?
- Does it aid the designer to avoid premature implementation details?

DESIGN EVALUATION CRITERIA-III

Safety:

- Can you ensure that the system will not enter an unsafe state?
- Can you ensure that you can always return to a safe termination state? (e.g., in the cruise control example, coasting down a steep hill is outside of the domain of the cruise control since only the throttle and not the brakes are controlled.)
- Is the model of the real—world within the system the same as outside the system? (e.g., is the definition of speed in the cruise control the same as a vehicle's actual speed?)

DESIGN EVALUATION CRITERIA-IV

Integration with hardware:

- How well do the modeled controls match the physical controls?
- How do the manual and automatic modes interact?
- What are the characteristics of real—time response? (How rapidly does the system react to control inputs? How accurately does the system maintain speed?)