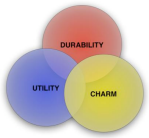
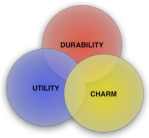


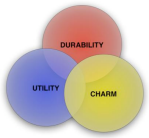

# Class 3 SSW565

Gregg Vesonder
Stevens Institute of Technology



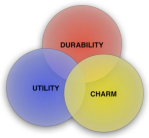

# Roadmap

- Logbook
- More on 4+1
- Architecture Styles
- Connectors
- Reading this week: "connector" paper, Mehta, Medivdovic & Phadke, 2000
- Readings next week: Sha, "Using simplicity to control complexity."



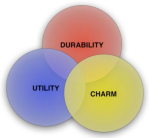

# Key Dates

- Thursday class May 28th
- Thursday class June 18th - MidTerm
- Thursday class June 25th
- Final July 20th

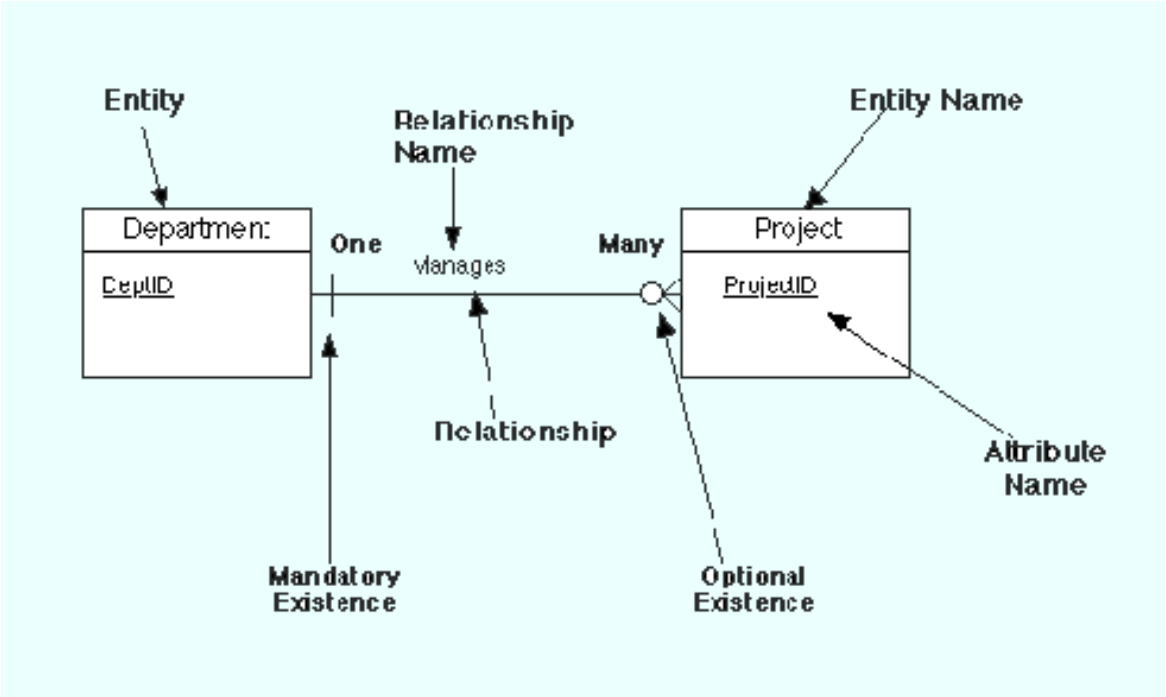



# Logbook

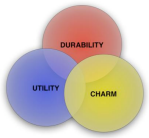
- In a bar: "We got a deal with the bank. They don't serve beer and we don't cash checks." - On modularity.



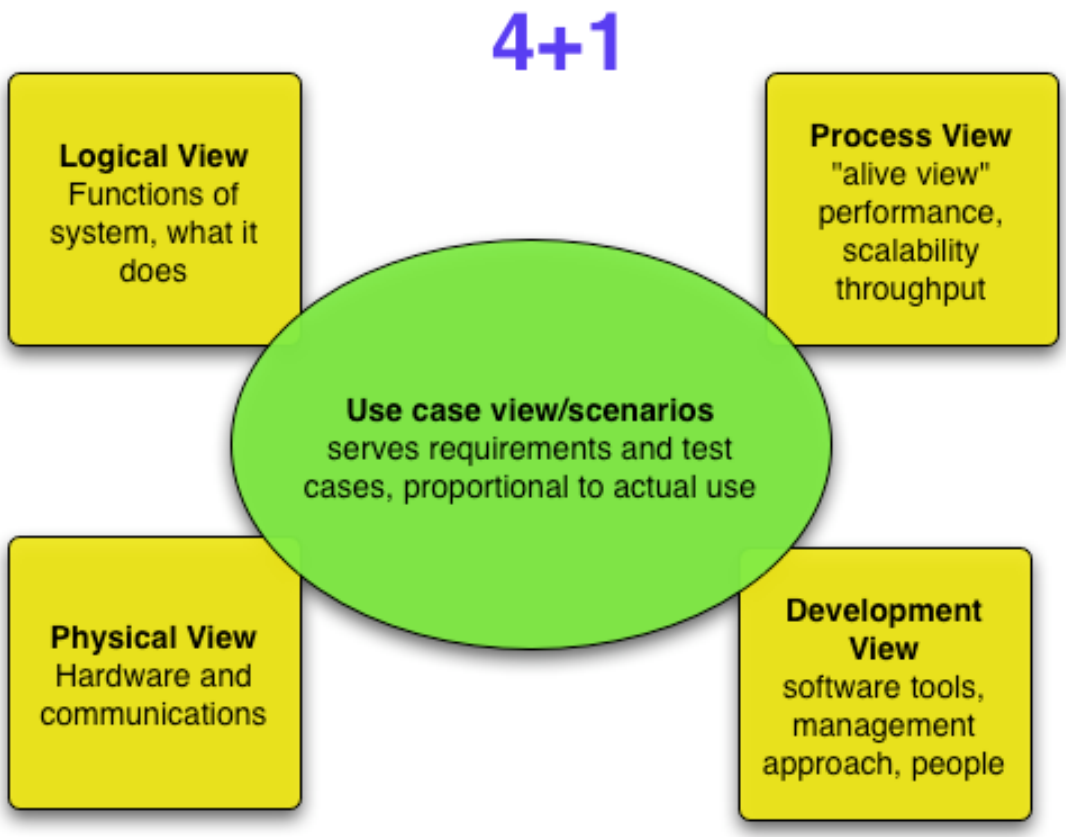

# Bonus Log

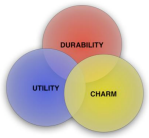



*http://www.utexas.edu/its-archive/windows/database/datamodeling/dm/erintro.html*




# Architecture Approach

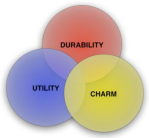





# An Example: Video Game

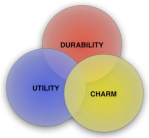
- A very simple Doom like video game - players walk through a maze, shooting software generated bad creatures with some intelligence. Sometimes these critters are called AI or bots.
- There can be more than one player playing and in this instance, the games are hosted on a centralized site.
- Let's call the game Gloom


*http://static.desktopnexus.com/wallpapers/7217-bigthumbnail.jpg*

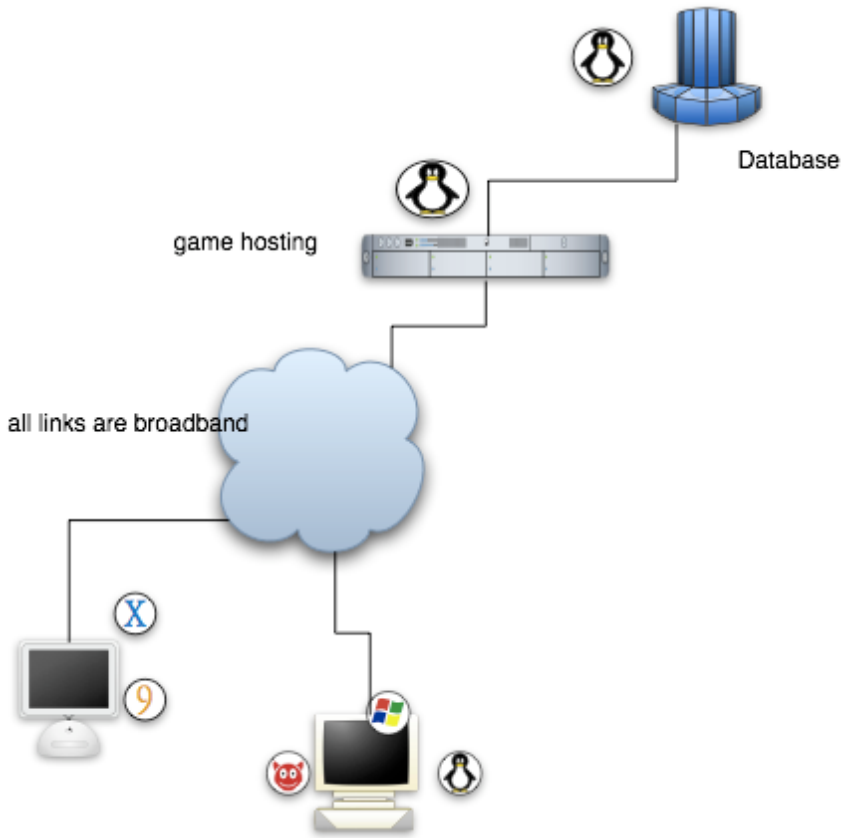


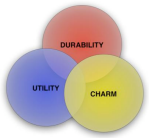

# Gloom Logical View

- No diagrams (and this is a bit ahead of things). The game itself is hosted using the Blackboard architectural style with each of the players and bots acting as knowledge sources
- The bots themselves gain their intelligence using a rule base architectural style.
- Different mazes and game results are stored in a repository, a relational data base.



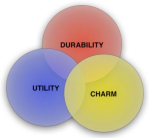

# Gloom Physical View





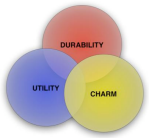

# Gloom Process View

- Each player is in its own Java sandbox.
- The bots for a given game session are one java sandbox and they communicate with a blackboard instance, also a separate process.
- Initial environment is uploaded from database and final scores are downloaded to database



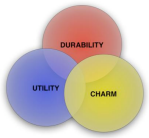

# Development View

- Java 1.4.2 is the Java version.
- Using an Oracle 9.0 database
- Blackboard software is done in house
- Clips is the rule based language used
- Since most of the coding is in java we will use Sun's coding conventions - http://java.sun.com/docs/codeconv/
- The team will use eXtremeProgramming



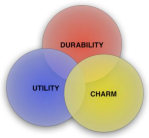

# Scenarios "+1"

- Single player: using a practice version solely against bots
- Tournament play: 2 players pitted against each other with brackets hosted on the game server
- Multiplayer environments with up to 12 players



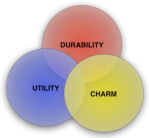

# On Architectural Styles

- Best we have at this moment
- Stimulates the gray cells to think about others
- Does not cover everything, not even close
- Most architectures are a mixture of these styles
- There are specialty areas not covered - feedback control
- The beginning of a long process
- Varied presentation here as appropriate
- Should get facile with these



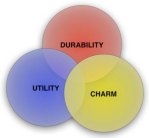

# Architectural Styles Taxonomy

- Independent Components
  - communicating processes
  - event systems
    - implicit invocation
    - explicit invocation
- Data Flow
  - batch sequential
  - pipes and filters
- Virtual Machine
  - interpreter
  - rule-based system
- Data Centered
  - repository
  - Blackboard (tuple space)
  - (hypertext)
- Call and Return
  - main program and subroutine
  - object oriented
  - Layered
- "New" Styles
  - Agents
  - Tiered (includes client/server)
  - Peer to Peer
  - Publisher-Subscriber
  - Distribution Tree
  - Beowulf derived

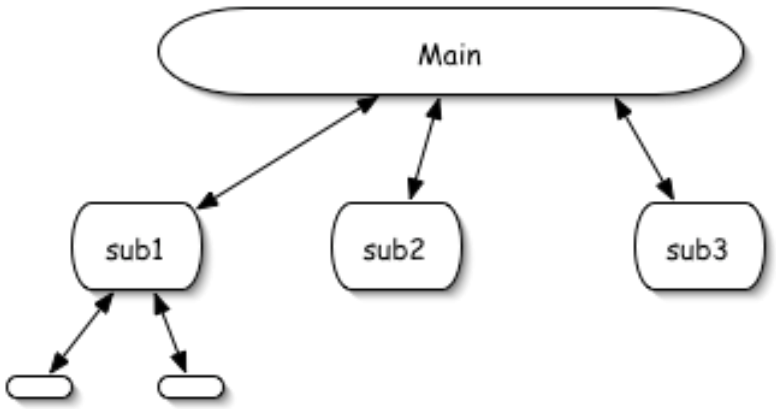



# On KWIC

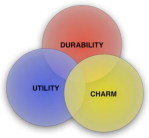
- Classic Parnas example, Key Word In Context
- Basically finding a word or phrase in a title
- input N lines title/line
- procedure
  - generate n shifts where n is # of words/line
  - store shifts in standard lexicographic (dictionary) order
  - output is KWIC index



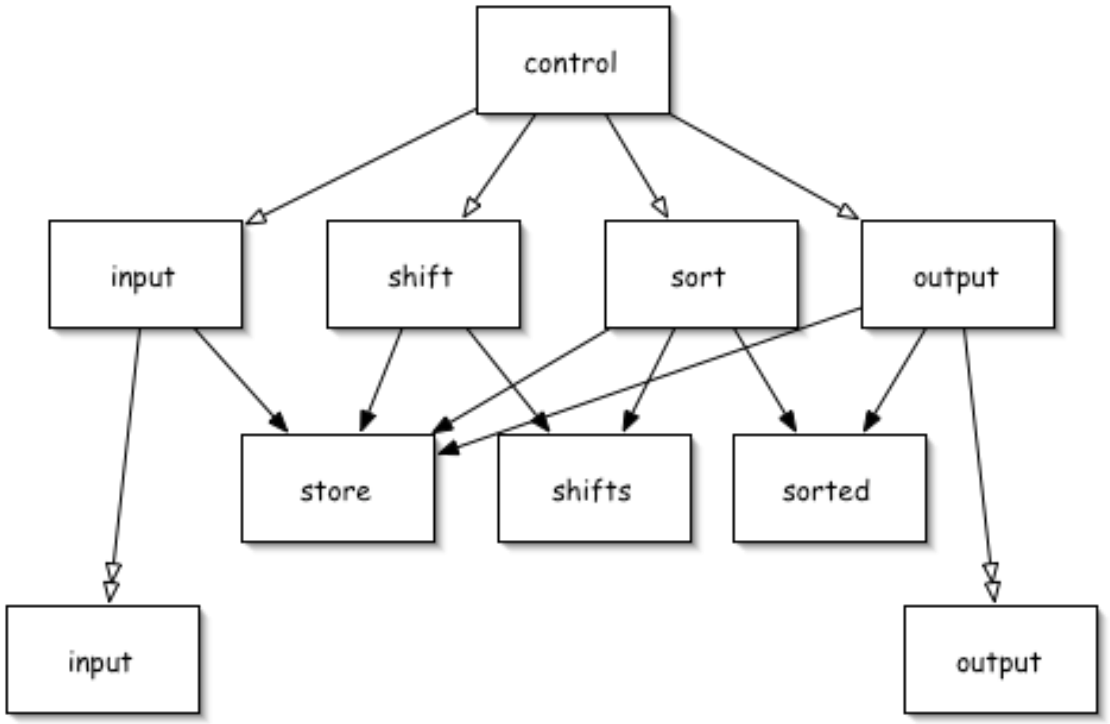

# Main Program with Subroutine



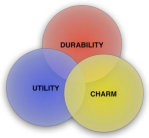

- Main program is controller
- Sequences through subroutines in a particular order



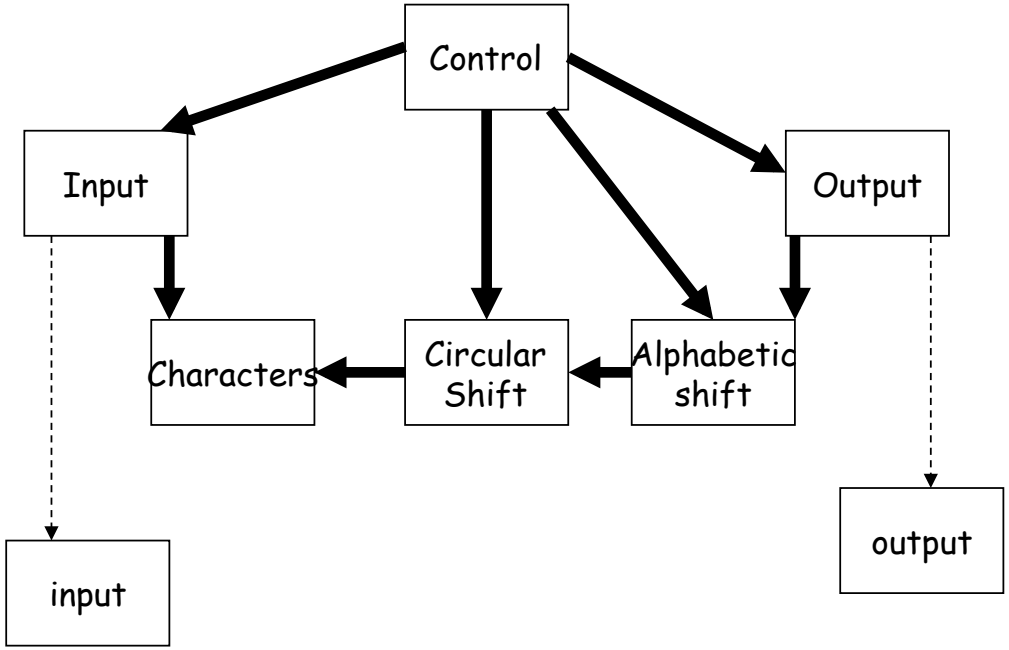

# Main Program with Sub (KWIC)




open-procedure call
closed-data access
double-i/o

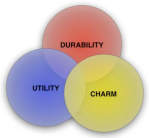



# Abstract Data Type (KWIC)

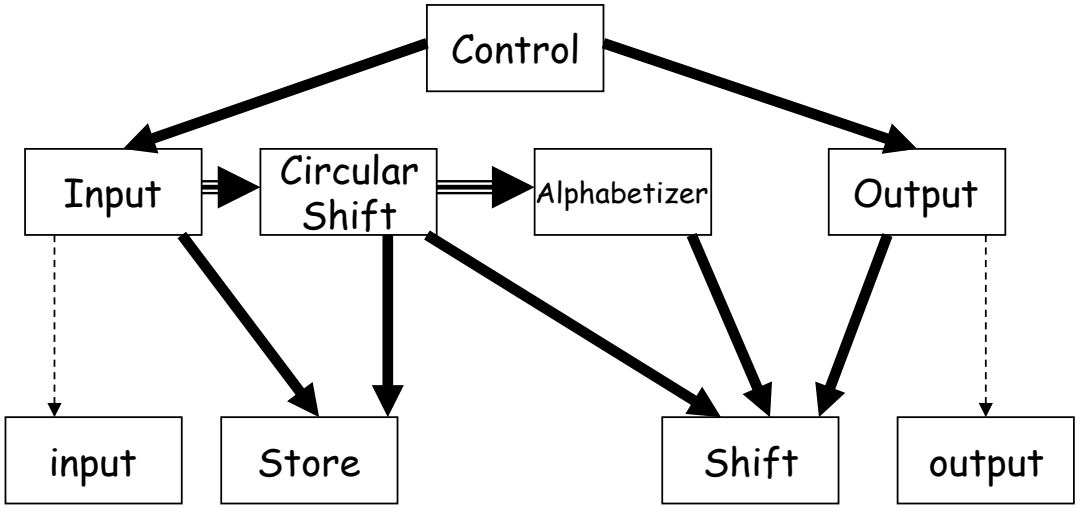



Solid - procedure call
Dashed - I/o

- Task decomposed into 5 modules
- Data is not directly shared
- Each module provides interface to access data
- Same logical decomposition as first but more amenable to both data and algorithm changes
- Reuse is supported since one module does not make assumptions about others
- BUT to add new functions, must modify existing modules (violating KISS) or add new modules with a potential performance penalty

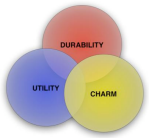



# Implicit Invocation (KWIC)

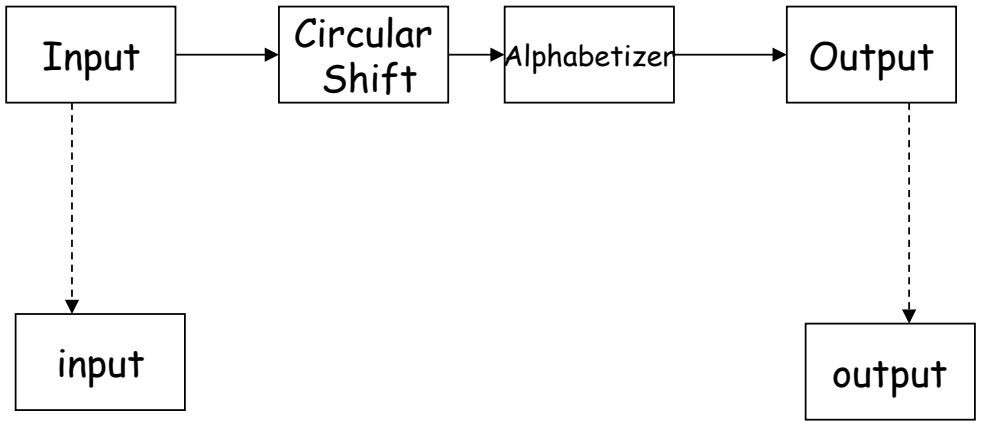



solid - procedure call
Striped- implicit invocation
dashed-i/o

- Uses shared data but access data abstractly
- Process of adding new line to store causes event to be sent to shift module, producing shifts that generates an event for the alphabetizer module (data driven)
- Additional modules can be attached and registered to be invoked on data changing events
- Since data is accessed abstractly, data can be changed easily too
- But difficult to control processing order of implicitly invoked modules

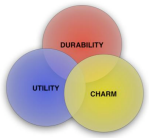



# Pipes and Filters (KWIC)

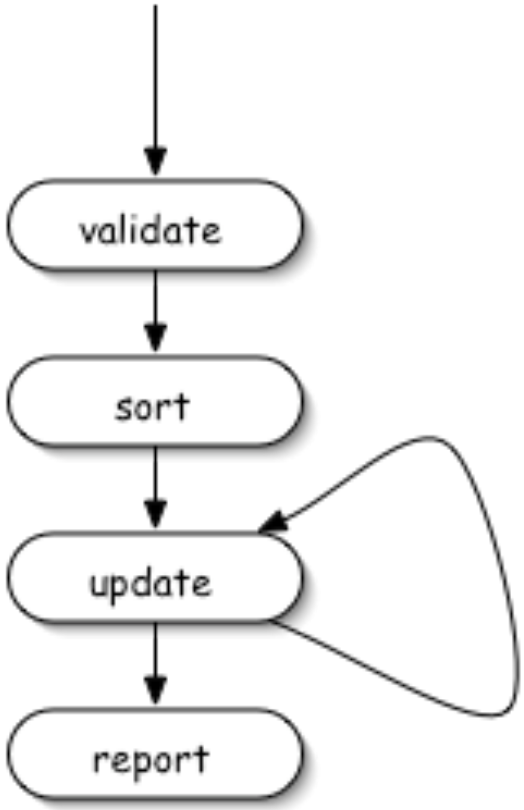



solid-pipe
dashed-i/o

- Maintains intuitive flow of processing
- Supports reuse, each filter can function in isolation
- Modification is easy, just add and subtract filters
- Difficult to support interactive system
- Poor use of space since each filter must copy all of the data to the output ports

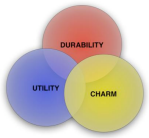



# Batch Sequential

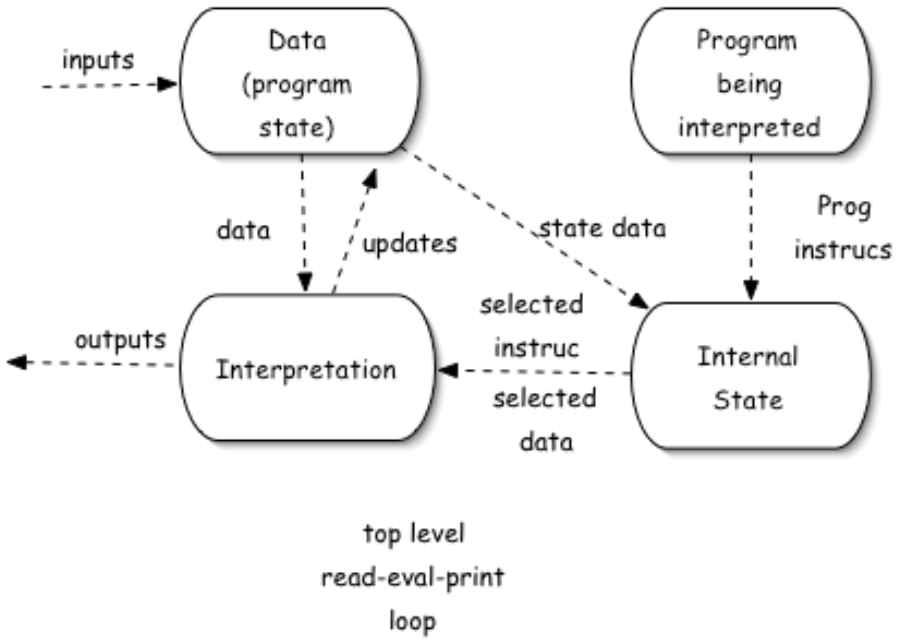



- Special form of pipe and filter but rather than processing a stream of data, each filter processes all the data as a whole and then moves on

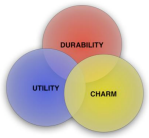
solid-tape



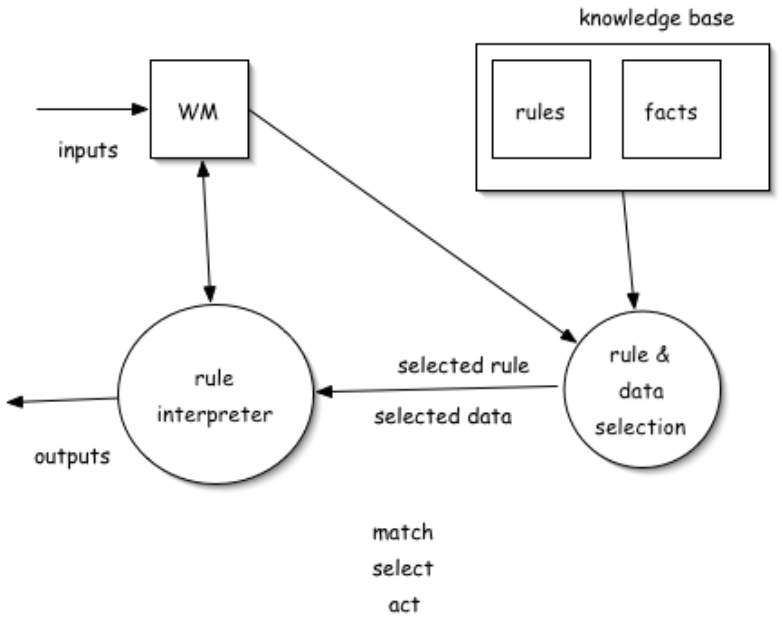

# Virtual Machine Style Interpreter



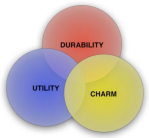

- For example JAVA and LISP -- same but different
- Portability



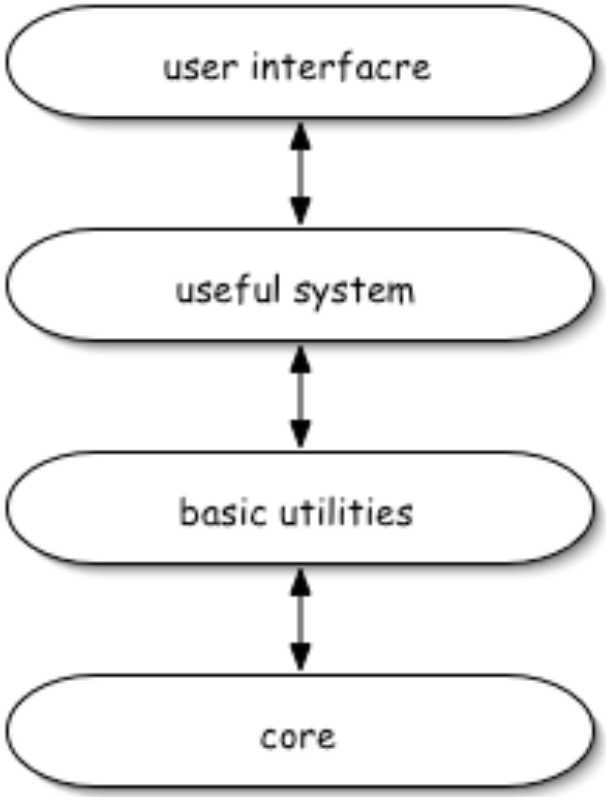

# Rule Based System



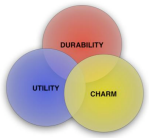

- Data driven applications
- Originally used for expert systems
- Can act as a blackboard WM blackboard, rules independent modules




# The Layered Style



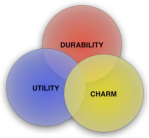

- Layered communication protocols, operating systems
- Supports layers of abstraction
- Supports enhancement, strongly controlled interaction
- Layers & their defined protocols support reuse
- Not every system can be pushed into a layered style
- Performance may be an issue and also finding the appropriate levels of abstraction




# What's AI

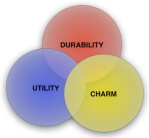
- Artificial Intelligence not only tries to understand intelligence but also build intelligent entities (Russell & Norvig )

| | | |
|---|---|---|
| Cognition | Systems that think like humans | Systems that think rationally |
| Behavior | Systems that act like humans -> empirical science | Systems that act rationally -> math and engineering |




# AI & Agents

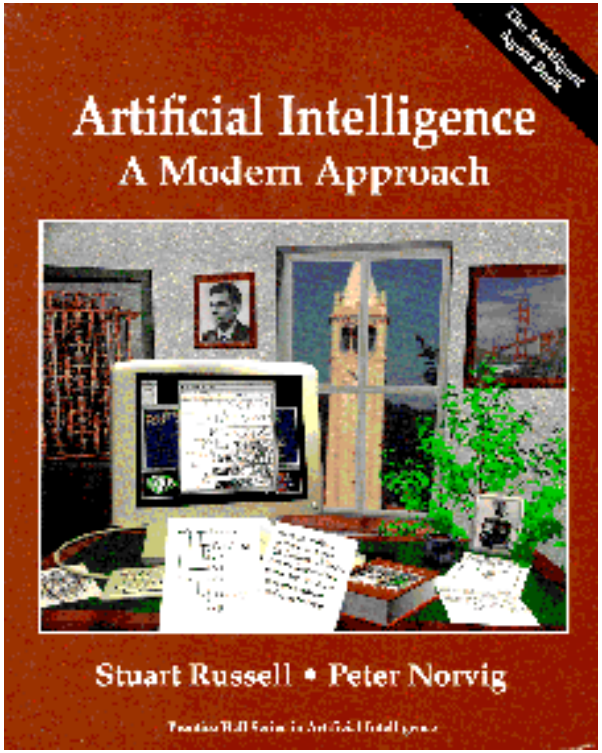
- Agent: something that acts, operates under autonomous control, perceiving the environment, persisting over long time periods, adapting, being able to take on another's goals
- A rational agent achieves the best outcome or, given uncertainty, the best expected outcome
- Perfect rationality - always doing the right thing, is not feasible in complicated environments
- Limited rationality - acting appropriately when there is not enough time (or ability or feasibility) to do all the calculations one might like

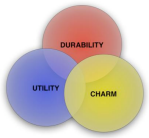



# Reference

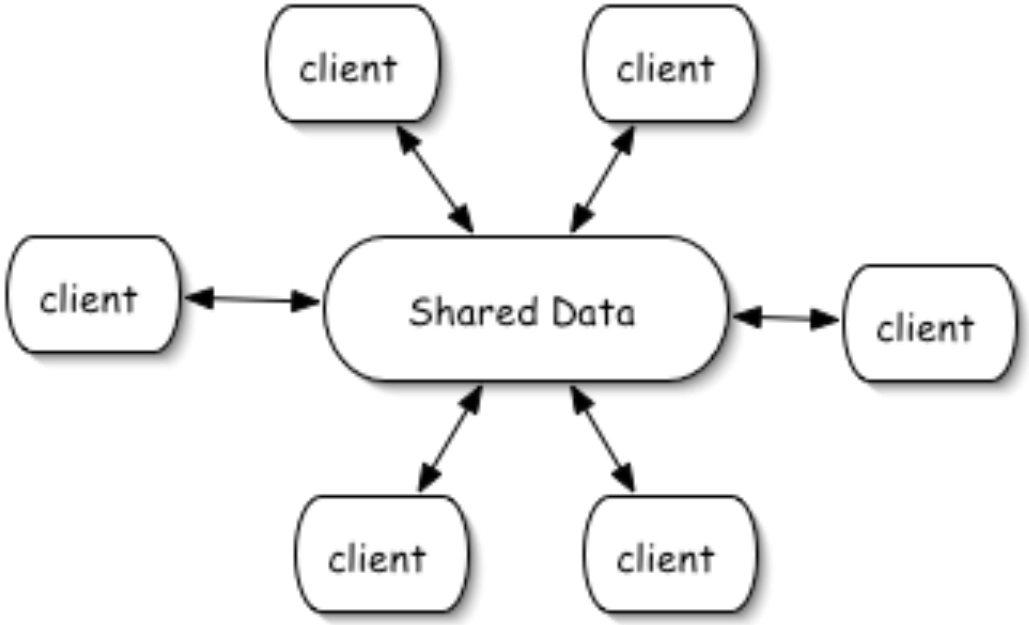
Russell & Norvig Artificial Intelligence: A Modern Approach 2nd Edition, Prentice Hall, ISBN: 0137903952 (now green cover)

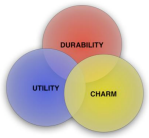
*More in a later lecture*



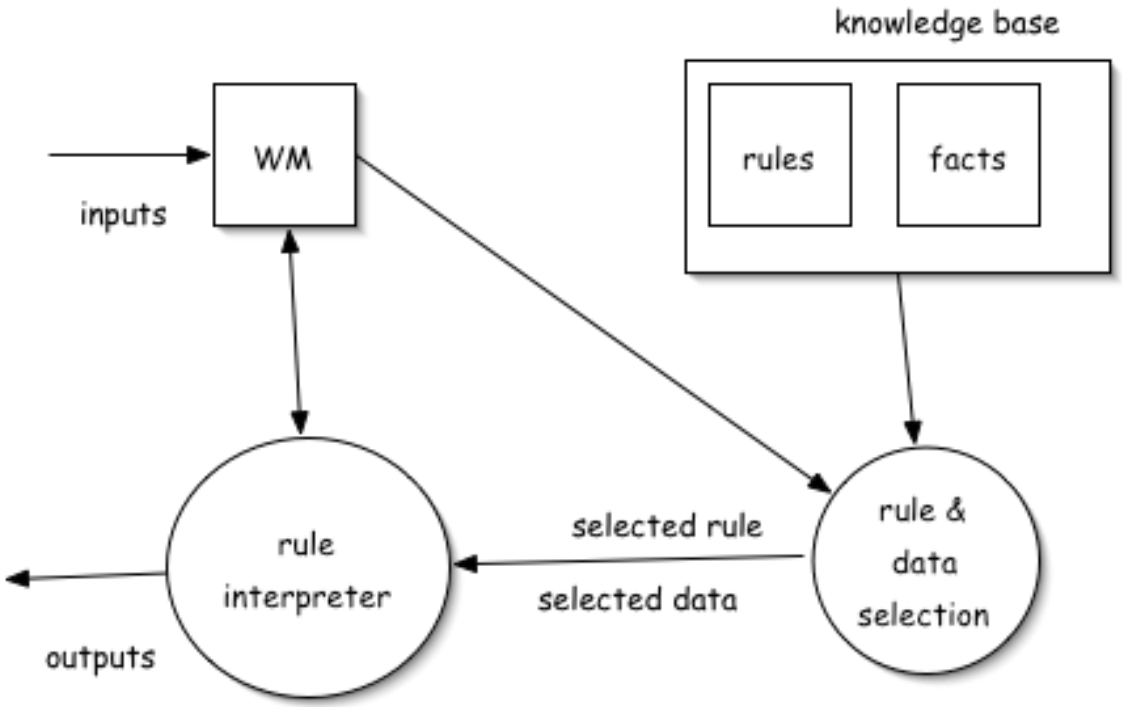



# Data-centered Style





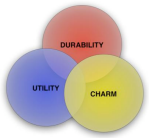

# Rule Based System



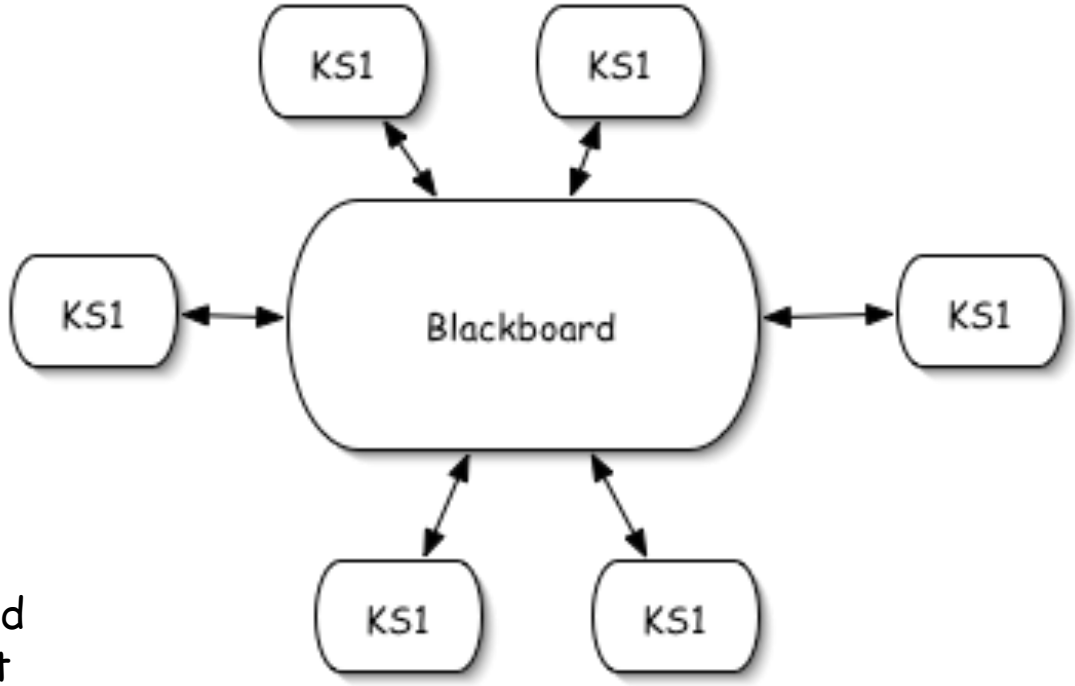

match

select

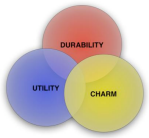
act




# Blackboard Model

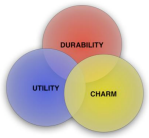



Note memory is blackboard and computation occurs at all independent KS elements



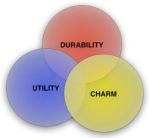

# Choosing Styles

- Data-flow- well defined inputs and outputs that are result of sequentially transforming the input in a time independent fashion
- Call and return-fixed order of computation and no ability to parallelize work must await results. Hierarchical.
  - OO-modifiability a key, reuse
  - layered-task divided into those specific to given app and those generic, portability important, reuse of other layered infrastructures
- Independent component - multiprocessor system, performance tuning is important (through reallocating work and processes to processors)



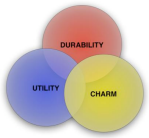

# Choosing Styles-2

- Communicating processes- message passing is sufficient, e.g.,
  - client-server/request-reply
  - heartbeat-overall state assessed and working in lock step
  - probe-echo-assess state of components (fault tolerant)
  - broadcast - time server
- Data-centered- issue is use and management of a large corpus of long-lived data
  - blackboard - AI systems, add layers of interpretation of data
- Virtual machine - no machine to run it on, but
  - data driven (rule based --- also blackboard)
  - portability, modifiability, transparency



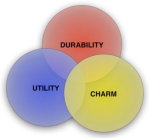

# More on Selection

| | main with sub | Abstract data | implicit | p&f |
|---|---|---|---|---|
| changes in data rep | - | + | + | - |
| changes in algorithm | - | o | o | + |
| changes in functionality | o | - | + | + |
| independent development | - | + | + | + |
| comprehensibility | -/+ | + | o | + |
| performance | + | + | - | - |
| reuse | - | + | + | + |



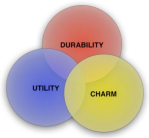

# 1, 2 & 3 Tier Models

- 1-tier, single machine, simplicity, easier to secure.Scalability is upgrading hardware (to a limited extent).
- 2-tier, client-server, variable division point, presentation, application (logic), data access. Issue of fat & thin clients
- 3-tier multi-tier, thin client- presentation, application (could be several layers), data access



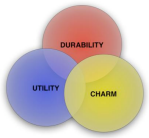

# Software Connectors

- Software connectors specify interaction among components in architectures and designs
- Manifestations of software connectors are buffers, procedure calls, networking protocols, pipes, shared variable access ...
- Connectors are key factors for system properties such as performance, resource use, scalability, reliability security, evolvability
- Architectures separate computation and storage (components) from interaction, transfer of control and data (connectors)
- Data and/or control are transfered along a duct, interaction channel with no associated behavior



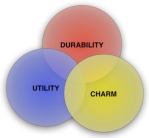

# Service Categories of Connectors

- communication- support transmission of data, e.g., stream, pipe
- coordination- support transfer of control, e.g., function calls and method invocations
- conversion -convert the interaction provided by one component to that required by another, e.g., adaptors
- facilitation- mediate and streamline component interaction



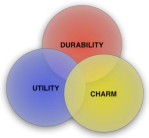

# Major Connector Types

- procedure call - model flow of control and transfer data, "assembly language of software interconnection"
- event - flow of control caused by an event, once it learns of the event all 'interested' parties are notified and control is yielded to components designated for processing these messages
- data access - permit components to access data maintained by a data store component
- linkage - enable duct formation and therefore tie system components and hold them in a state during operation -required to grow, monitor and repair existing systems

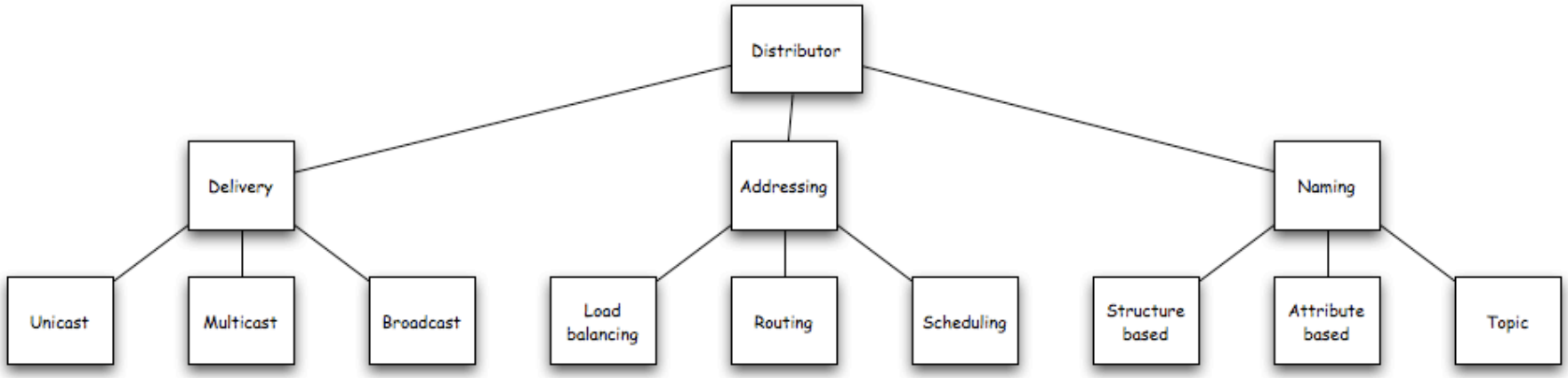



# Connector Types - II

- streams - transfer large amounts of data between autonomous processes, often represent connectors with complex protocols of usage (UNIX pipes, TCP/UDP sockets)
- arbitrator - streamline system operation and resolve conflicts, assure system trustworthiness and dependability, negotiate service levels
- adaptor - support interaction between components that were not designed to interoperate
- distributor - identify interaction paths and routing of communication and coordination information, always work in conjunction with other connectors, e.g., DNS. Affects scalability and survivability




# First A Gentle Diagram



# From Mehta, Medivdovic & Phadke, 2000

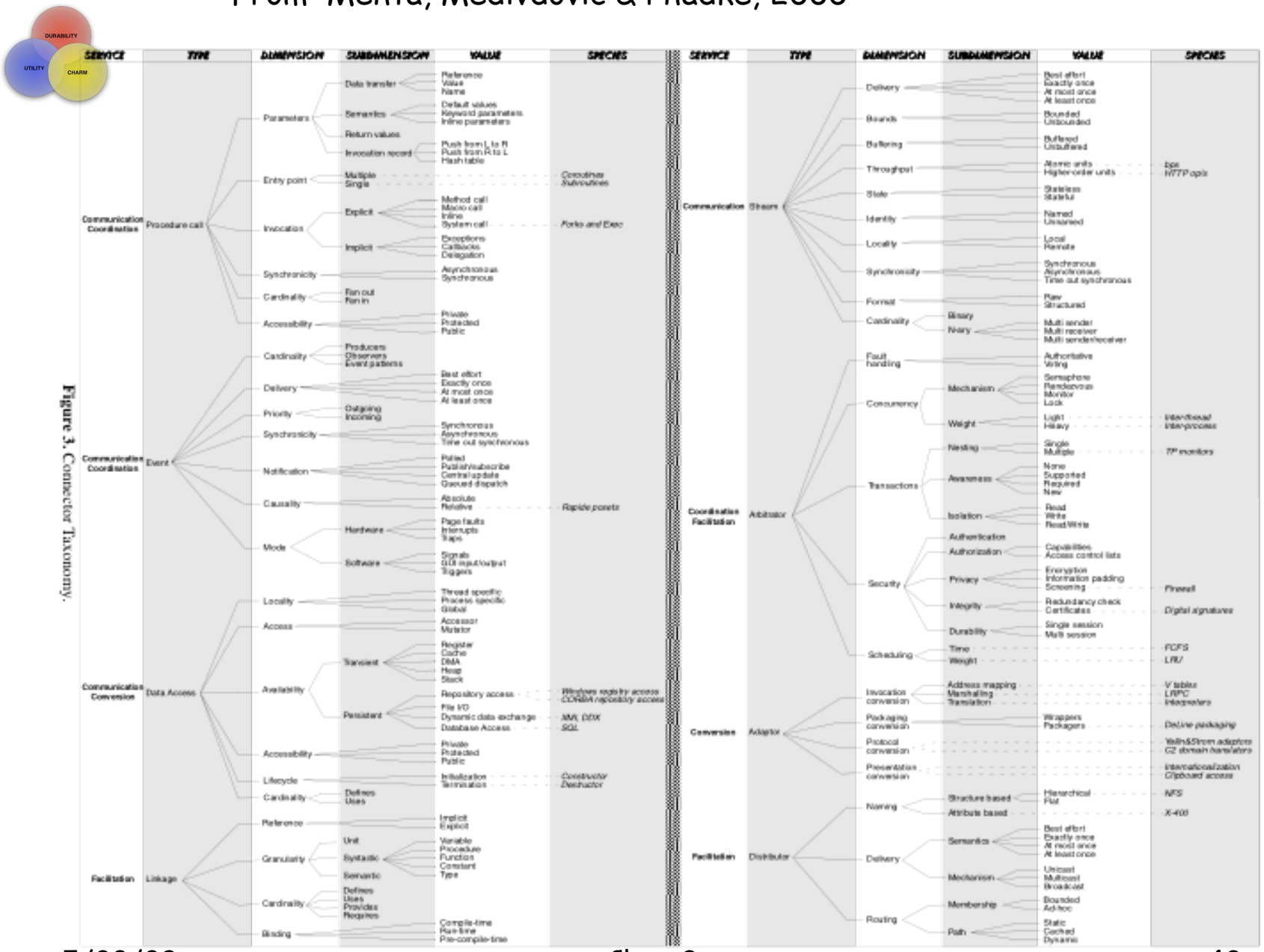
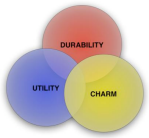
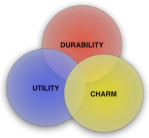



Figure 3. Connector Taxonomy.




*What is the mapping between connector types and architectural styles?*




# Other References

- Shaw, M and Garlan, D., Software Architecture, 1996, Prentice Hall.