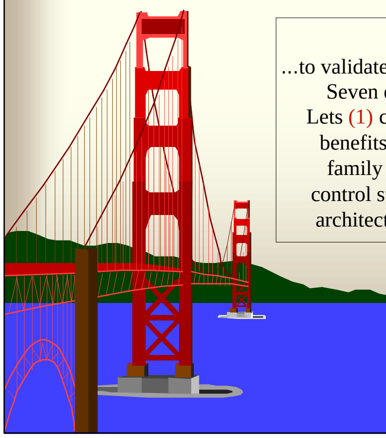
# Software Architecture

## Perspectives on an Emerging Discipline

### CS 531 SW Requirements Specification and Analysis

**Chapter Three**



**Learning Objective**

...to validate the *Software Architectural Styles* given in the prior chapter. Seven examples illustrate how we can use architectural principles. Lets (1) consider how solutions to the same problems provide different benefits; (2) get experience developing a domain specific style for a family of industrial products; (3) find out how to apply a process-control style to system design; and (4) see examples of heterogeneous architectures to understand why they were architected in such a way.

Frederick T Sheldon
Assistant Professor of Computer Science
*University of Colorado at Colorado Springs*

# *Chapter Three*

## Case Studies in Architectural Design

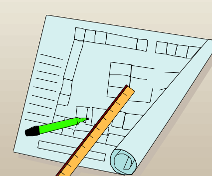
Criteria for decomposing a system into modules (components or sub-units)

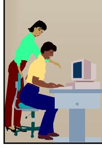
- Functional with shared access to data (representations)
- Hiding design decisions

---

PREMISE: Different problem decomposition strategies vary greatly in their ability to withstand design changes … changes in the

- Algorithm,
- Data representation,
- Enhancements to system functions,
- Performance (time and space) and
- Reuse





# Four Architectural Designs for the KWIC System

See page 33 for a description of KWIC

KWIC

Input / Shift / Alphabetize / Output

Lets consider the issues of algorithm, data representation, enhancements to system functions, performance (time/space) and reuse... against several solution types.





# Sol-1: Main/Subprogram w/ Shared Data

See Figure 3.1

Advantages:

- Data is represented and used efficiently
- Intuitive appeal

Disadvantages:

- Change in data storage format affects all modules

# Sol-2: Abstract Data Types

See Figure 3.2

## Same five modules but no data sharing:

Each module provides an interface that permits other components to access data only by invoking procedures in that interface

Same logical decomposition as Main/Subprogram

## Advantages:

Both algorithms and data can be changed without affecting the other modules





# Sol-2: Abstract Data Types

Continued

## Advantages (continued)

Reuse is supported:

- Modules make fewer assumptions about the others

## Disadvantages

Not well suited for certain kinds of functional enhancements

- Modifying existing modules may compromise their simplicity **or**
- Adding new modules may lead to performance penalties

# Sol-3: Implicit Invocation

See Figure 3.3

Uses shared data except for *two* important differences...

Interface to the data is *more abstract* (using list or set but they do not expose storage formats)

Interactions are based on an *active data* model

- E.g., the act of adding a “new line” to the line storage causes an event → the shift module.
  - (1) circular shifts (in a separate, abstract-data store) and,
  - (2) the alphabetize is then implicitly invoked.




# Sol-3: Implicit Invocation

Continued

## Advantages

Supports functional enhancement...

- Additional modules can be registered so that they will be invoked by *data changing events*.

Insulates computations (data is accessed abstractly) from changes in data representation

Supports reuse

- Implicitly invoked modules rely only on the *existence* of certain externally triggered events → de-couples modules from each other!

# Sol-3: Implicit Invocation

Continued

## Disadvantages

Invocations are data driven and therefore:

- Difficult to control the processing order
- Most natural implementations of this kind tend to use more space!





# Sol-4: Pipes and Filters

See Figure 3.4

Uses four filters working in a sequence

Control is distributed

Filters run when input data is available

No data sharing except the piped data stream

Desirable properties (advantages)

Maintains the intuitive flow of processing

Supports reuse

- Each filter can function in isolation
- New functions easily added by inserting filters at the appropriate points in the processing sequence

## Sol-4: Pipes and Filters

Continued

- Disadvantages
  - Virtually impossible to modify the design to support interactive system
  - Deleting a line would some persistent shared storage
    - Violates a basic tenet of this approach
  - Uses space inefficiently
    - Filters copy all the to its input port



## Comparisons

- Shared data supports
  - Change in function and performance
- ADTs supports
  - Change in data representation, performance and reuse
- Implicit invocation supports
  - Change in algorithm and function
- Pipe and filter supports
  - Reuse and change in function + algorithm

## Comparisons Must be Cognizant of Certain Design Considerations

Intended use

- Batch versus interactive
- Update intensive versus query-intensive
- For example: Pipes and filter solution
  - Easily allows insertion of new filters (supports changes in algorithm, function and reuse) but the data representation is *wired* into assumptions about the kind of data that is transmitted along the pipes!
  - Additional overhead may also involve the parsing and un-parsing of the data into pipes




## Instrumentation Software
## Case Study

Purpose: develop a reusable system architecture for oscilloscopes

- Functionality and features
  - Performs dozens of measurements
  - Megabytes of internal storage
  - Interface to a network of workstations & instruments
  - Sophisticated user interface:
    - Touch panel screen
    - Built-in help facilities
    - Color displays

# The Problems

- Legacy of heterogeneous conventions and programming languages across the company
- Rapidly changing market demands
- Need to meet the demands of specialized markets
  - General purpose → patient monitoring → automotive diagnostics
- Performance was suffering because
  - Different operational modes were satisfied by loading different software which was getting larger and larger




# The Goals and Results

- Develop and architectural framework that would address these problems
- Results:
  - Domain specific SW architecture as a basis for the next generation of oscilloscopes
  - The framework has been extended and adapted to accommodate a broader class of systems
  - Also, refined to better suit the needs of the instrumentation software!

## 1st Attempt: Object-Oriented Model

See Fig. 3.6

Clarified the data types

Waveform, signals, measurements, trigger modes, …

However, this fell short of expectations due to:

- No overall model that explained how the types fit together
- Not clear how to partition functionality
  - Should the measurements be associated with the types of data being measured,
  - Which objects should the UI interface with





## 2nd Attempt: A Layered Model

See Fig. 3.7

Core layer - signal manipulation functions

Filter signals as they enter the oscilloscope

Subsequent layers

Waveform acquisition (2), manipulation (3)

- Measurement, addition of waveforms, Fourier transformation . . .

Display functions (4)

- Mapping digitized waveforms and measurements to visual representations
- Responsible for interacting with the user

## 2nd Attempt: A Layered Model → Debacle!

Layered models was intuitively appealing

Unfortunately. . .

Wrong model for the application domain!

- The boundaries of abstraction enforced by layers conflicted with the needs for interaction among various functions!
- Model suggested that all interactions with the user should be with the visual representation…but,
- Real users need to directly affect the functions at all layers
  - E.g., setting attenuation at the signal manipulation layer, choosing acquisition mode and parameters at the acquisition layer, etc.)





## 3rd Attempt: Pipe and Filter Model

See Fig. 3.8

Functions viewed as incremental transformers of data

- Signal transformers used to condition external signals
- Acquisition transformers derive digitized waveforms from these signals
- Display transformers convert these waveforms into visual data

## Significant Improvement, *Except...*

Functions were not isolated in separate partitions

- Nothing prevents signal data from feeding directly into display filters
- Model was intuitive wrt the engineers view of signal processing
  - Allowed clean intermingling and substitution of HW and SW components within a system design!
- However, one main *problem* was
  - *Unclear* how the user would interact with it!!!
  - User put simply at visual end → worse than layered model!





## 4th Attempt: Modified Pipe and Filter Model

See Fig. 3.9

Accounts for the user inputs by associating with each filter a control interface

- Setting the sample rate, configuration parameters
- The filters were modeled as *higher order* (HO) functions
- The HO functions determine what data transformation the filter will perform

## Solved the UI Problem

- Provided a collection of settings to dynamically modify aspects of the oscilloscope characteristics
  - Decoupled certain functions from the UI (as was needed for the signal processing functions)
  - UI can treat the signal processing functions solely in terms of the control parameters
    - Changes in the implementation SW/HW are possible with out affecting the implementation of the UI




## *Further Specialization Yet*

- Still, performance was poor
  - Each time  a filter needs to process a wave-form it copies a significantly large chunk of internal storage!
  - Further, different filters run at radically different speeds!  … potential to cause a significant bottleneck
- Solution was to introduce colors of pipes
  - Some allowed processing w/o copying
  - Some allowed incoming data to be ignored
  - In all, tailoring of pipe/filter computations

# What Have We Learned

- Seen some *real* issues
- Emphasized the *trade-offs*
- See that typical industrial SW must be *adapted* to the specific domains
- The final result depended greatly on the properties of the pipe and filter architectures



# Mobile Robotics System
## Case Study

- Controls manned/partially manned vehicles
  - Space exploration, hazardous waste disposal, underwater exploration
- The software must deal with:
  - External sensors and actuators
  - Real-time responsiveness
  - Acquire sensor I/P, control motion and plan future paths
  - Many issues from *imperfect* inputs to *unexpected/unpredictable* obstacles, and events

# Design Considerations: *Mobile Robots*

**Four basic requirements….**

**Accommodate deliberate and reactive behavior**

- Coordinate actions it must undertake to achieve its designated objective (collect rock sample, avoid obstacles)

**Allow for uncertainty**

- Framework for actions even when faced with incomplete or unreliable information (contradictory sensor readings)

**Account for dangers**

- Must be fault tolerant, safe and with high performance (e.g., cope with reduced power, dangerous vapors, etc.)

**Give design flexibility**

- Development requires frequent experimentation and reconfiguration



# Lets Examine Four Architectural Designs

Lozano's Control Loops

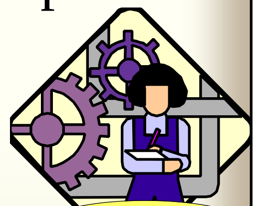
Elfes's Layered Organization

Simmons's Task Control Architecture

Shafer's Application of Blackboards

# Solution 1: Control Loop

***— Specifics —***



**See Fig. 3.9**

Industrial robots need only handle minimally unpredictable events

- Tasks are fully defined (no need for a planer) and has no responsibility wrt its environment
  - Open loop paradigm applies
  - Robot initiates actions without caring about consequences
- Lets add feedback for closed loop
  - Robot adjusts the future plans based on monitored information



# Solution 1: Control Loop

*Requirements Trade-Off Analysis*

Req1– Advantage: simplicity

- Simplicity is a drawback in more unpredictable environments
- Robots mostly confronted with disparate discrete events that require them to switch between *very* different behavior modes
- For complex tasks, gives no leverage for decomposition into cooperating components

Req2– Advantage: reducing unknowns through iteration

- Is biased toward one method (only).
- Trial and error process of action-reaction to eliminate possibilities at each turn.
- No framework for integrating these with the basic loop or for delegating them to separate entities.

# Solution 1: Control Loop

*Requirements Trade-Off Analysis Continued*

Req3– Advantage: supports fault tolerance and safety

- Simplicity makes duplication easy
  - Reduces the chance of errors creeping into the system

Req4 – Advantage: clearly partition-able into supervisor, sensors and motors that are independent and replaceable

- More *refined tuning* is however not really supported (inside the modules)

Conclusion: *Most appropriate for simple robotic systems*




# Solution 2: Layered Architecture

***— Specifics —***

Influenced the sonar and navigational systems design used on the Terregator and Neptune mobile Robots…

- Level 1 (core) control routines (motors, joints,..),
- Level 2-3 real world I/P (sensor interpretation and integration (analysis of combined I/Ps)
- Level 4 maintains the real world model for robot
- Level 5 manage navigation
- Level 6-7 Schedule & plan robot actions (including exception handling and replanning)
- Top level deals with UI and overall superviosory functions

# Solution 2: Layered Architecture

*Requirements Trade-Off Analysis*

Req1 Avoids some problems encountered in the control loop style by defining more components to delegate tasks.

Defines abstraction levels (robot control versus navigation) to guide the design

Does not however fit the actual data / control flow patterns!!





# Solution 2: Layered Architecture

*Requirements Trade-Off Analysis (continued)*

Information exchange is less straightforward because the layers suggest that *services* and *requests* be passed between layers

- Fast reaction times drives the need to bypass layers to go directly to the problem-handling agent at level 7 *…skip layers to improve response time!*

Two separate abstractions are needed that are not supported

- Data hierarchy
- Control hierarchy

# Solution 2: Layered Architecture

*Requirements Trade-Off Analysis (continued)*

Req2 Abstraction layers address the need to manage uncertainty

What is uncertain at the lower layers may become clear with added knowledge available from the higher layers

For Example

- The context embodied in the world model can provide the clues to disambiguate conflicting sensor data



# Solution 2: Layered Architecture

*Requirements Trade-Off Analysis (continued)*

Req3 Fault tolerance and passive safety (strive not to do something)

Thumbs up data and commands are analyzed from different perspectives

Possible to incorporate many checks and balances

Performance and active safety may require that layers be short circuited

## Solution 2: Layered Architecture

*Requirements Trade-Off Analysis (continued)*

Req4 Flexibility in replacement and addition of components

- Interlayer dependencies are an obstacle
- Complex relationships between layers can become more difficult to decipher with each change

Success because the layers provide precision in defining the roles of each layer




## Solution 3: Implicit Invocation

*Basis and Specifics*

- Based on various hierarchies of tasks
  - Utilizes dynamic task trees
    - Run-time configurable
    - Permits selective concurrency
- Supports 3 different functions
  - Exceptions
    - Suited to handle spontaneous events
    - Manipulate task trees
  - Wiretapping
    - Messages intercepted by tasks superimposed on a task tree
  - Monitors
    - Read info and execute some actions if data fulfills a criterion

# Solution 3: Implicit Invocation

*Requirements Trade-Off Analysis*

Req1 Advantage: clear cut separation of action

- Explicit incorporation of concurrent agents in its model

Req2 Disadvantage: uncertainty not well addressed

- Task tree could be built by exception handler

Req3 Advantage: accounts for performance, safety, & fault tolerances

- Redundant fault handlers
- Multiple requests handled concurrently





# Solution 3: Implicit Invocation

*Requirements Trade-Off Analysis (continued)*

Req3 advantage: Accounts for performance, safety, & fault tolerances

- Redundant fault handlers
- Multiple requests handled concurrently

Req4 advantage: Incremental development & replacement straightforward

- Possible to use wiretaps, monitors, or new handlers without affecting existing components

# Solution 4: Blackboard Arch.

*Basis and Specifics*

- Based on CODGER system used in NAVLAB project (known as *whiteboard arch)*
- Relies on abstractions similar to those found in the layered architecture example
- Utilizes a shared repository for communication between components


Univ. of Colorado at Colorado Springs


# Solution 4: Blackboard Arch.

*Basis and Specifics (continued)*

- Components register interest in certain types of data
  - This info is returned immediately or when it is inserted onto blackboard by some other module
- Components of CODGER architecture are:
  - Captain: overall supervisor
  - Map navigator: high-level path planner
  - Lookout: monitors environment for landmarks
  - Pilot: low-level path planner and motor controller
  - Perception subsystems: accept sensor input and integrate it into a coherent situation interpretation


Univ. of Colorado at Colorado Springs

## Solution 4: Blackboard Arch.

*Requirements Trade-Off Analysis*

Req1 Deliberative and reactive

- Components register for the type of information they are interested in and receive it as it becomes available
- This shared communication mechanism supports both deliberative and reactive behavior requirements
- However, the control flow must be worked around the database mechanism; rather than communication between components



## Solution 4: Blackboard Arch.

*Requirements Trade-Off Analysis*

Req2 Allow for uncertainty

- provides means for resolving conflicts or uncertainties as all data is in database (from all components)
- modules responsible for resolution simply register for required data and process it accordingly

## Solution 4: Blackboard Arch.

*Requirements Trade-Off Analysis (continued)*

Req3 - account for environment dangers

- Separate modules that watch the database for unexpected situations provide exception mechanism, monitoring and wiretapping to adjust for environment conditions and deliver safety, reliability, reaction time guarantees, etc.





## Solution 4: Blackboard Arch.

*Requirements Trade-Off Analysis (continued)*

Req4 - design flexibility

- Maintenance is facilitated by de-coupling senders from receivers
- Component concurrency is supported
- There is some loss of design flexibility due to the fact that control is intrinsically dependant on shared database

## Solution 4: Blackboard Arch.

*Summary*

- Capable of modeling cooperation of tasks
  - task coordination
  - flexible resolution of uncertainty
- Based on implicit invocation mechanism triggered by shared database contents
- Workable solution that is slightly less powerful than the TCA Implicit Invocation solution




## Comparisons *The Score Card*

*Task coordination*
Ctl loop 2 | Layers 0 | ImpInvoc 3 | Blkbrd 1

*Dealing with uncertainty*
Ctl loop 0 | Layers 2 | ImpInvoc 2 | Blkbrd 1

*Fault-tolerance*
Ctl loop 2 | Layers 2 | ImpInvoc 3 | Blkbrd 1

*Safety*
Ctl loop 2 | Layers 2 | ImpInvoc 3 | Blkbrd 1

*Performance*
Ctl loop 2 | Layers 2 | ImpInvoc 3 | Blkbrd 1

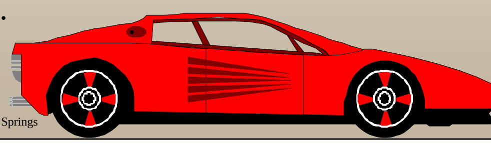
*Flexibility*
Ctl loop 2 | Layers 0 | ImpInvoc 1 | Blkbrd 1

# Cruise Control Case Study

Illustrates the application of the control-loop paradigm to a simple problem that is traditionally cast in object oriented terms.

Demonstrates that the use of the control-loop architecture can contribute significantly to clarifying the important architectural dimensions of the problem.

This case study has traditionally been used by Booch and others to compare/contrast object oriented and functional programming.







# Problem Statement

## According to Booch

See Fig. 3.16

A cruise control (CC) system that exists to maintain the constant vehicle speed even over varying terrain.

***Inputs:***

| | |
|---|---|
| System On/Off: | If on, maintain speed |
| Engine On/Off: | If on, engine is on. CC is active only in this state |
| Wheel Pulses: | One pulse from every wheel revolution |
| Accelerator: | Indication of how far accelerator is de-pressed |
| Brake: | If on, temp revert cruise control to manual mode |
| Inc/Dec Speed: | If on, increase/decrease maintained speed |
| Resume Speed: | If on, resume last maintained speed |
| Clock: | Timing pulses every millisecond |

***Outputs:***

| | |
|---|---|
| Throttle: | Digital value for engine throttle setting |

# Some Issues with Booch's Problem Statement

Ambiguity about rules for deriving the O/P from the I/Ps

Ambiguity about what speed is to be controlled

- Current speed versus maintained speed

Stated output is a throttle setting value versus a *change* in throttle setting (as expected in classical process control)

- *Change* output avoids calibration + sensor wear problems

Specifies a millisecond clock used in combination with wheel pulses to compute current speed over specified ...

- A slower clock or one that delivered current (precise) time on demand would work while requiring less computing resources.


Univ. of Colorado at Colorado Springs



# Restatement of Cruise-Control Problem

*Whenever the system is active, determine the desired speed, and control the engine throttle setting to maintain that speed.*


Univ. of Colorado at Colorado Springs

# Booch's Object View of Cruise Control

See Fig. 3.17

Each element corresponds to important quantities and physical entities in the system

Each blob represents objects

Each directed line represents dependencies among the objects




# Process-Control View of Cruise Cntl

Appropriate for SW embedded in a physical system that involves continuing behavior:

- Especially for systems subject to external perturbations
  - True in the case of cruise control

Essential System Elements:

- Computational Elements
  - Process definition - take throttle setting as I/P & control vehicle speed
    - Details irrelevant - while driving a mechanical device controlled by 1 or more computers.
  - Control algorithm - current speed (wheel pulses) compared to desired speed
    - Change throttle setting accordingly presents the issue:
      - decide how much to change setting for a given discrepancy

# Essential System Elements
(continued)

Essential System Elements:

- Control algorithm
  - model current speed from wheel pulses
  - compare to desired speed
  - change throttle setting accordingly (decide how much to change throttle setting for a given discrepancy)

Data Elements

- Controlled variable: current speed of vehicle
- Manipulated variable: throttle setting
- Set point: set by accelerator and increase/decrease speed inputs
  - system on/off, engine on/off, brake and resume inputs also have a bearing
- Controlled variable sensor:
  - modeled on data from wheel pulses and clock



# Control-loop View Sub-Problems

See Fig. 3.19

"Whenever the system is active determine the desired speed."

How to determine if the system is active given that there are a variety of events that trigger the system

Use a state machine to determine active/inactive state of system

how to compute desired speed

The desired speed (set point) is the current speed as modeled from wheel pulses and increase/decrease speed controls

# Control-loop View Sub-Problems

(continued)

See Fig. 3.18

"Control the engine throttle setting to maintain that speed."

How to model the current speed from wheel pulses

Model could fail if the wheel spins

- Wheel pulses from spinning drive wheel - cruise control maintaining wheel speed (at constant speed) even if vehicle stops
- Wheel pulse from non-drive wheel with spinning drive wheel - cruise control will act as if current speed is too slow and continually increase throttle setting

What control authority does the process have

Brake is not under control of the process (only throttle)

If vehicle coasts faster than desired speed, the controller cannot slow it down





# Complete Cruise Control System

See Fig. 3.21

Combines control architecture, state machine and event table to form system

System represents all of Booch's objects with clear roles

Design strategy for this system could easily be hybrid

Employ control-loop architecture for the system as a whole

Employ one or more other architectures (including objects and state machines) to elaborate the elements of the control loop architecture

# Control-Loop Summary

Shift from Object Oriented to Control-Loop (CL) raised a number of important design issues

Limitations of the CL model also became clear:

- Possible inaccuracies in the current speed model
- Incomplete control at speed higher than set point

Data flow character of the CL model exposed irregularities in the way input is specified to the system

- mixture of state and event inputs
- inappropriateness of the absolute position of the accelerator





# Analysis and Discussion

Control View Clarified Design:

- Led to re-specify the O/P as the actual speed of the vehicle (current speed)
- Separating control from process makes speed model explicit and therefore more likely to be validated
  - Also raised the question of control authority
- Explicit control algorithm elements sets up design decision about the kind of control the be exercised
- Establishing relationships among components control paradigm discriminates among different kinds of inputs and makes the feedback loop more obvious
- Clearly separates manual and automatic operation modes
- Set point determination is easier to verify when separated from control

# Control-Loop Methodologies

A methodology should help the designer to:

- Decide when an architecture is appropriate
- Identify the elements of the design and their interactions
- Identify critical design decisions and potential safety problems
- Provide for system modification

Astron and Wittenmark Methodology      Choose:

- Control principle, control variables, the measured variables, and ...
- Create appropriate subsystems




# Performance: System Response to Control

Process control provides powerful tools for the selection and analysis of the response characteristics of the system

Example: Cruise-controller can set throttle in several ways:

- On/Off control: simple on/off control of process (more applicable systems like a thermostat)
  - hysteresis could be used to control fluttering of power
- Proportional control: the output is a fixed multiple of the measured error
  - can lead to steady state values that are not quite equal to the set point or to oscillation around the set point
- Proportional plus Rest control: a proportional response to the error in combination with an ever changing output as long as the error is present
  - tends to force error towards zero
  - can speed correction by basing on a derivative of error speeds (probably over kill for cruise control)

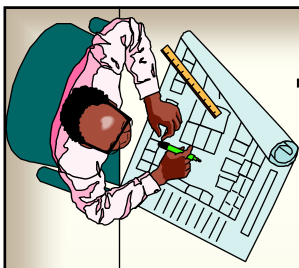
# Summary

Much of the design methodology expressiveness arises from how well it focuses attention on the significant decisions at appropriate times

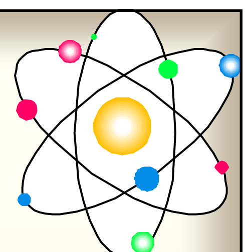
In the cruise-control example, higher level decisions were better elicited by the methodology based on *process control* than for the more common *object oriented* methodology

- Control paradigm separates the operation of the main process from compensation for external disturbances
- Yielded appropriate abstractions
- Revealed important design issues



# Three Vignettes in Mixed Style



Purpose is to review three systems with mixed styles of architecture

- PROVOX process control system
- Hayes-Roth Rule Based system
- HEARSAY II speech recognition system

# PROVOX

The Fisher Controls PROVOX system offers distributed process control for chemical production processes:

- simple control loops to control pressure, flow, levels
- complex strategies involving interrelated control loops
- provisions for integration with plant management and information systems





# PROVOX 5 Level Hierarchy

See Fig. 3.22

Integrates process control with plant management and other corporate information systems

- Level 1: Process measurement and control
  - direct adjustment of final control elements
- Level 2: Process supervision
  - operations console for monitoring and controlling Level 1
- Level 3: Process management
  - computer based plant automation; including management reports, optimization strategies, and guidance to the operations console
- Level 4 & 5: Plant and corporate management
  - higher level functions such as cost accounting, inventory control, and order processing/scheduling

# PROVOX Architecture

Different computation and response times are required at the different levels of the system

- Therefore different computation models are used to achieve these results
  - Levels 1 - 3: object-oriented
  - Levels 4 - 5: Largely based on conventional data processing repository (database) models



# Process-Control (Levels 1 - 3)

PROVOX uses a set of points (or Loci)

**See Fig. 3.23**

- seven specialized forms support the most common kinds of control
- point are object-oriented design elements
- points encapsulate information about control points of the process
- points are individually configured to achieve desired control strategy
- Data associated with the points includes:
  - Operating Parameters
    - current process value
    - set point (target value)
    - valve output
    - mode (automatic or manual)

# Process-Control (Levels 1 - 3)

(continued)

Data associated with the points includes (continued):

- Tuning Parameters
  - Gain, reset, derivative and alarm trip points
- Configuration Parameters
  - Tag (name), I/O channels

Points can include a template for a control strategy

Points include procedural definitions such as:

- Control algorithms and communication connections
- Reporting services and trace facilities




# Process-Control (Levels 1 - 3)

(continued)

Collections of points implement the desired process-control strategy

Through the communication services, and

Through the actual dynamics of the process (example: One point increasing the flow in a tank will be reflected in another point that senses tank level)

Reports from points appear as input transactions to the data collection and analysis processes at higher design levels

The process designer can organize points into:

Control processes

Processes can be aggregated into Plant Process Areas and Plant Management areas

# Plant/Corporate Management (Levels 4 -5)

Provisions for integration with plant management and business systems exist at Levels 4 and 5

- Provides transaction to these systems (typically selected independently from process control system)
- Systems are commonly designed as database repositories with transaction processing functions supporting a central data store (as opposed to the object-oriented design seen in the lower levels)
- Hierarchical design at the top levels:
  - Permits strong separation of different classes of functions, and
  - Clean interfaces between the layers; but
  - Are often too intricate to permit strict layering




# Rule Based Systems

See Fig. 3.24

Provide a means of codifying the problem solving skills of human experts

- Captured as sets of situation-action rules
- Whose execution or activation is sequenced in response to the conditions of the computation (rather than being predetermined)

Hayes-Roth rendering of a Rule Based system includes:

- Pseudocode - to be executed
- Interpretation engine - rule interpreter (heart of interface engine)
- Control state of interpretation engine - rule and data element selector
- Current state of program - working memory

# Expanded Hayes-Roth Rule Based System

See Fig. 3.25 and 3.26

Rule based systems heavily use pattern matching and context (currently relevant rules)

Added special mechanisms to facilitate these features complicate the original simple interpreter design

Combining figures 3.24 and 3.25 simplifies the resulting model and leads to the following:

Knowledge base is a relatively simple structure; yet is able to distinguish between active and inactive components

Rule interpreter is implemented as a table driven interpreter

- With control procedures for pseudocode and
- Execution stack modeling the current program state





# Expanded Hayes-Roth Rule Based System (continued)

"Rule and data element selection" is implemented as a pipeline

- that progressively transforms active rules and facts to prioritized activations
- the third filter ("nominators") uses a fixed database of meta-rules

Working memory is not further elaborated

# Hayes-Roth Conclusions

In a sophisticated rule-based system, elements of the simple rule-based system are elaborated in response to the execution characteristics of the particular class of languages being interpreted

- Retains the original concept to guide understanding and ….
- Ease later maintenance of the system

As the design is elaborated, different components can be elaborated with different idioms

Rule-based model can itself be thought of as a design structure:

- Set of rules whose control relations are determined during execution by computation state
- A rule-based system provides a virtual machine (rule extractor) to support this model





# Blackboard Globally Recast as an Interpreter

See Fig. 3.27

Blackboard model of problem solving is a highly structured special case of opportunistic problem solving

- Solution space is organized into several application dependent hierarchies
- Domain knowledge is partitioned into independent modules of knowledge that operate on knowledge within and between levels

HEARSY-II speech recognition system was the first major blackboard architecture system

- Implemented between 1971 and 1976 on DEC PDP-10
- 6 to 8 level hierarchy in which each level abstracts information from its adjacent lower level
- Blackboard elements represent hypotheses about the interpretation of an utterance

# HEARSAY-II
## (continued)

Knowledge sources correspond to tasks like:

- segmenting the raw signal
- identifying phenomes
- generating word candidates
- hypothesizing syntactic segments
- proposing semantic interpretations

Knowledge sources contain:

Condition Part:

- specifies when it is appropriate

Action Part:

- process relevant blackboard elements and generate new ones



# HEARSAY-II
## (continued)

See Fig. 3.28 and 3.29

Control component is realized as a blackboard monitor and scheduler

scheduler monitors blackboard and calculates priorities for applying knowledge source to various blackboard elements

PDP-10 was not directly capable of condition-triggered control

HEARSAY-II implementation compensates by providing mechanisms of a virtual machine to realize implicit invocation semantics

this addition complicates Fig 3.27

Blackboard model can be recovered by

suppressing the control mechanism and

regrouping the conditions and action into knowledge sources

# HEARSAY-II
## (continued)

Function assignment facilitates the virtual machine in the form of an interpreter

- Blackboard corresponds cleanly to the current state of the recognition task
- Collection of knowledge sources roughly supply the pseudocode of the interpreter
  - actions also contribute
- Interpretation engine includes:
  - blackboard monitor
  - focus-of-control database
  - scheduler
  - actions and knowledge sources



Univ. of Colorado at Colorado Springs

# HEARSAY-II
## (continued)

- Scheduling queue corresponds to control state
- Condition contribute to rule selection as well as forming pseudocode
  - to the extent that the condition execution determines priorities



Univ. of Colorado at Colorado Springs

# HEARSAY-II Conclusions

- System initially designed with one model (blackboard, a special form of repository)
- System realized through a different model (interpreter)
- Interpreter view invokes a different aggregation of components that the blackboard view
  - as opposed to a component by component expansion as in the previous two examples