Lecture 3

Software Architectures
Roadmap of the course

- What is software architecture?
- Designing Software Architecture
  - Requirements: quality attributes or qualities
  - How to achieve requirements: tactics
  - How do tactics lead to architectural styles
- Today: case studies on architectural styles
  - to also observe the achieved qualities
Common Architectural Idioms

- **Data flow systems**
  - Batch sequential
  - Pipes and filters
- **Call-and-return systems**
  - Main program & subroutines
  - Hierarchical layers
  - OO systems
- **Virtual machines**
  - Interpreters
  - Rule-based systems
- **Independent components**
  - Communicating processes
  - Event systems
- **Data-centered systems (repositories)**
  - Databases
  - Blackboards
1. KWIC

- In his paper (1972) David Parnas proposed the following problem

The KWIC (Key Word in Context) index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

*(On the Criteria to be Used in Decomposing Systems into Modules - discusses the important design decisions that impact how we modularize our software systems)*
Architectural solutions for KWIC

- Shared data
- Abstract Data Types
- Implicit invocation
- Pipe-and-filter
KWIC design issues

- Changes in the processing algorithm
- Changes in the data representation
- Enhancement to system function
- Performance
- Reuse
Main Program/Subroutine with shared data

- Problem decomposed according to 4 basic functions
  - Input, shift, alphabetize, output
- These components coordinated by main program that sequences through them
- Data in shared storage
- Communication: unconstrained read-write protocol
  - Coordinator ensures sequential access to data
KWIC – shared data solution

FIGURE 3.1  KWIC: Shared Data Solution
Shared data, pro and cons

- Advantages
  - Data can be represented efficiently
  - Intuitive appeal

- Disadvantages
  - Modifiability
    - Change in data format affects all components
    - Change in overall processing algorithm
    - Enhancements to system function
    - Reuse not easy to do
Abstract data types (ADT)

- Similar set of five modules, with interfaces
- Data is not shared by computational components
  - Accessed via interfaces
KWIC, ADT solution
ADT, pro and cons

- Advantages
  - Logical decomposition into processing modules similar to shared data
  - Algorithms/data can be changed in individual modules w/o affecting others
  - Better reuse (module has fewer assumptions about other modules)

- Disadvantages
  - Enhancing the function
    - Modify existing modules -> bad for simplicity, integrity
    - Add new modules -> performance penalties
Implicit invocation

- Shared data as the integration mechanism
- More abstract data interfaces
  - Data accessed as a list/set
- Computations invoked implicitly when data is modified
  - Line added -> event to shift module
  - Circular shifts produced in another shared data store -> event to alphabetizer, invoked
KWIC, implicit invocations
Implicit invocation, pro and cons

• Advantages
  • Functional enhancements easily
  • Data changes possible
  • Reuse

• Disadvantages
  • Difficult to ctrl processing order of implicitly invoked modules
  • Data representation uses more space
Pipe and filter

- Four filters
  - Input, shift, alphabetize, output
  - Process data and send it to the next
- Distributed ctrl
- Data sharing
  - Only the one transmitted on pipes
Pipe and filter, pros and cons

• Advantages
  • Maintains intuitive flow of processing
  • Reuse supported
  • New functions easily added
  • Amenable to modifications

• Disadvantages
  • Impossible to modify design to get interactive system
  • Data is copied between filters — space used inefficiently
KWIC, pipes and filters
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Shared data</th>
<th>ADT</th>
<th>Impl. invocation</th>
<th>Pipe and filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes in the processing algorithm</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Changes in the data representation</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enhancement to system function</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Performance</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reuse</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
2. Instrumentation software

- Develop a reusable system architecture for oscilloscopes
  - Rely on digital technology
  - Have quite complex software
- Reuse across different oscilloscope products
  - Tailor a general-purpose instrument to a specific set of users
- Performance important
  - Rapid configuration of software within the instrument

=> Domain-specific software architecture
Object-oriented model of software domain

- Clarified the data types used for oscilloscopes
  - Waveforms, signals, measurement, trigger modes, …
- No overall model to explain how the types fit together
- Confusion about partitioning of functionality
  - Should measurements be associated with types of data being measured or represented externally?
  - Which objects should the user interface interact with?
Layered model of the oscilloscope

- Well-defined grouping of functions
- Wrong model for the application domain
  - Layer boundaries conflicted with the needs of the interaction among functions
    - The model suggest user interaction only via Visualization, but in practice this interaction affects all layers (setting parameters, etc)
Pipe-and-filter model

- Oscilloscope functions were viewed as incremental transformers of data
- Corresponds well with the engineers’ view of signal processing as a dataflow problem
- Main problem:
  - How should the user interact?
Modified pipe-and-filter model

- Each filter was associated with a control interface
  - Provides a collection of settings to be modified dynamically by the user
  - Explains how the user can make incremental adjustments to SW
  - Decouples signal-processing from user interface
- Signal-processing SW and HW can be changed without affecting the user interface as long as the control interface remains the same
Modified pipe-and-filter model, more

- Further specialization
  - Pipe-and-filter lead to poor performance
    - Problems with internal storage and data exchange between filters
    - Waveforms have large internal storage \(\Rightarrow\) not practical for filters to copy waveforms every time they process them
    - Filters may run at radically different speeds
      - Not good to slow faster filter just to keep the pace with slower ones
  - Solution: several types of pipes (distinct colours)
    - Some allowed data processing w/o copying
    - Slow filters allowed to ignore incoming data when already processing other data
    - \(\Rightarrow\) the pipe/filter computations more tailorable
Instrumentation software summary

- Case study shows
  - Some issues for developing architectures for industrial SW
  - Different styles => different effects on solution
- Software must be typically adapted from pure forms to specialized styles (domain specific)
- Here the result depended on properties of pipe-and-filter architecture adapted to satisfy the needs of the product family
3. Mobile Robotics

- The system controls a manned or partially manned vehicle
  - Car, submarine, space vehicle, …
- Build software to control the robot
  - External sensors and actuators
  - Real-time
- Input provided by sensors
- Control the motion
- Plan the future path
Mobile Robotics con’t

- Complicating factors
  - Obstacles may block the path
  - Imperfect sensor input
  - Robot might run out of power
  - Accuracy in movement
  - Manipulation with hazardous material
  - Unpredictable events might lead to need of rapid response
Mobile Robotics con’t

- Consider four (4) architectural designs
  - Control loop
  - Layered design
  - Implicit invocation
  - Blackboard
Mobile Robotics con’t

- **Design considerations**
  - **Req 1: deliberative and reactive behaviour**
    - Coordinate robot actions with environment reactions
  - **Req 2: uncertainty**
    - The robot needs to act based on incomplete and unreliable information
  - **Req 3: account for dangers**
    - Fault tolerance, safety, performance
  - **Req 4: flexibility**
    - Application development requires experimentation and reconfiguration
Mobile Robotics con’t

- Requirements of different kind, application depends on complexity and predictability
  - Robot in another planet $\Rightarrow$ fault tolerance
- The four requirements guide the evaluation of the four architectural alternatives
Solution 1: control loop

- A mobile robot uses a closed-loop paradigm
  - The controller initiates robot actions and monitors their consequences, adjusting plans
Solution 1 con’t

• The four requirements?
  • Req 1: deliberative and reactive behaviour
    • + simplicity of paradigm
    • - simplicity a problem in unpredictable environments
      • Implicit assumption: continuous changes in environment require continuous reaction
      • Robots face discrete events
      • Switch between behaviour modes - how to change between modes?
    • How to decompose the software into cooperating components?
Solution 1 con’t

- The four requirements?
  - Req 2: uncertainty
    - A trial-and-error process
  - Req 3: account for dangers
    - Simplicity makes duplication easy
  - Req 4: flexibility
    - The major components (supervisor, sensors, motors) separate and replaceable
Solution 1 con’t

• Summary:
  • Paradigm appropriate for simple robotics
  • Can handle only a small number of external events
  • No really for complex decomposition of tasks
Solution 2: layered architecture

- Eight (8) levels:
  - Level 1: Robot control routines (motors, joints, ...)
  - Levels 2&3: input from the environment
    - Sensor interpretation and integration
  - Level 4: robot’s model of the real world
  - Level 5: navigation
  - Levels 6&7: scheduling and planning of robot actions
  - Level 8: user interface and supervisory functions
Solution 2 con’t

• The four requirements?
  • Req 1: deliberative and reactive behaviour
    • + More components to delegate tasks
    • + indicates concerns that must be addressed
    • + defines abstraction levels to guide the design
    • - does not fit the data and control-flow patterns
    • - does not separate the data hierarchy from the control hierarchy
Solution 2 con’t

• The four requirements?
  • Req 2: uncertainty
    • + abstraction layers manage this
  • Req 3: account for danger
    • + managed by the abstraction mechanism: data and commands are analysed from different perspectives
      • Fault tolerance and passive safety ok; active safety not ok
  • Req 4: flexibility
    • - interlayer dependencies an obstacle
    • - complex relationships between layers can become difficult to manage
Solution 2 con’t

• Summary:
  • Provides a framework for organizing components
    • Precise about roles of layers
  • Problems when adding detail at implementation level
    • The communication pattern in a robot will not follow the scheme of the architecture
Solution 3: implicit invocation

- Task-control architecture
  - Based on hierarchies of tasks
    - Task trees
    - Parent tasks initiate child tasks
    - Software designer can define temporal dependencies between tasks
    - Dynamic reconfiguration of task trees at run time
  - Uses implicit invocation to coordinate interaction between tasks
    - Tasks communicate by multicasting messages via a message server
Solution 3 con’t

- Task-control architecture supports:
  - Exceptions: exception handling override tasks
    - Change processing mode
    - Can abort or retry tasks
  - Wiretapping: intercept messages by superimposed tasks
    - Safety-checks of outgoing commands
  - Monitors: read information and execute actions
    - Fault-tolerance issues using agents to supervise the system
Solution 3 con’t

- The four requirements?
  - Req 1: deliberative and reactive behaviour
    - + Separation of action and reaction via the task trees and exceptions, wiretapping and monitors
    - + concurrency explicit: multiple actions can proceed simultaneously and independently
      - - though in practice limited by the central message server
      - - relies on a central control point
Solution 3 con’t

- The four requirements?
  - Req 2: uncertainty
    - not explicitly in the model
      - Maybe via task trees and exceptions
  - Req 3: dangers
    - exception, wiretapping, monitors
    - fault tolerance by redundancy
      - Multiple handlers registered for same signal concurrently
  - Req 4: flexibility
    - implicit invocation allows incremental development and replacement of components
      - Often sufficient to register new handlers in central control
Solution 3 con’t

- Summary:
  - TCA offers a comprehensive set of features for coordinating tasks
  - Appropriate for complex robot projects
Solution 4: blackboard

- Based on the following components:
  - **Captain**: overall supervisor
  - **Map navigator**: high-level path planner
  - **Lookout**: monitors the environment
  - **Pilot**: low-level path planner and motion controller
  - **Perception subsystem**: input from sensors and integration the input to an interpretation
Solution 4 con’t

• The four requirements?
  • Req 1: deliberative and reactive behaviour
    • + components interact via the shared repository
    • - control flow must be coerced to fit the database mechanism
      • Components do not communicate directly
  • Req 2: uncertainty
    • + blackboard the means for resolving conflicts and uncertainties
      • All data available in the database
Solution 4 con’t

- The four requirements?
  - Req 3: account for dangers
    - + communication via a central service, the database
      - Exception handling, wiretapping, monitors can be implemented by adding modules that watch the database for certain signs of problematic situations
  - Req 4: flexibility
    - + Supports concurrency
    - + Decouples senders from receivers
      - Facilitates maintenance
Solution 4 con’t

- Summary:
  - The architecture is capable of modelling the cooperation of tasks
    - Coordination
    - Resolving uncertainty
  - Slightly less powerful than TCA
  - Not the only possibilities for robotics …
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Control loop</th>
<th>Layers</th>
<th>Impl. invocation</th>
<th>Blackboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task coordination</td>
<td>+ -</td>
<td>-</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Dealing with uncertainty</td>
<td>-</td>
<td>+ -</td>
<td>+ -</td>
<td>+</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>+ -</td>
<td>+ -</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Safety</td>
<td>+ -</td>
<td>+ -</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Performance</td>
<td>+ -</td>
<td>+ -</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Flexibility</td>
<td>+ -</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
4. Cruise Control

• The control loop paradigm applied to a problem traditionally seen in OO-eyes
• The control-loop architecture clarifies the architectural aspects of the problem
• Previously used to explore differences between OO and procedural programming
Cruise control con’t

• A cruise-control system maintains the speed of a car, even over varying terrain.

• Inputs:
  • System on/off
  • Engine on/off
  • Pulses from the wheel
  • Accelerator
  • Brake
  • Increase/decrease speed
  • Resume speed
  • Clock

• Output
  • throttle
Cruise control con’t

• How to derive output from the inputs?
• Inputs provide two kinds of information:
  • Is the cruise control on?
    • If yes, what speed should be maintained?
• Output is a value for the engine throttle setting
  • The corresponding signal should change the throttle setting
  • A more conventional cruise-control would specify control of current speed
    • Current speed here only implicitly as maintained speed
Cruise control con’t

• A millisecond clock
  • Used in combination with wheel pulses to determine the current speed
  • The process that computes the speed will count the number of clock pulses between wheel pulses
    • The problem is over specified
    • A single system clock is not required
Cruise control con’t

• Restatement of the problem:
  • Whenever the system is active, determine the desired speed and control the engine throttle setting to maintain that speed
Solution 1: OO view

- An OO decomposition is arranged around objects that exist in the task description
  - Correspond to quantities and physical entities in the system
    - Blobs - objects
    - Lines - dependencies among objects
  - Desired speed appears here as the target speed
    - Not explicitly present in the original problem statement
Process-control paradigm

- Continuous processes convert input materials to product
- Values of measurable properties of system state constitute the **variables of the process**
  - Not to be confused with program variables
- Process variables that measure the output materials are called **controlled variables** of the process
- **Manipulated variables** are associated with things that can be changed by the control system to regulate the process
Process-control paradigm con’t

- Definitions
  - Process variables
  - Controlled variables
  - Input variables
  - Manipulated variables
  - Set point
  - Open-loop
  - Closed-loop
  - Feedback control system
  - Feedforward control system
Process-control paradigm con’t

- The purpose of a control system is to maintain specified properties of the outputs of the process at given reference values called set points
Software paradigm for control systems

- An architectural style that controls continuous processes can be based on the process-control loop:
  - Computational elements:
    - Process definition
    - Control algorithm
  - Data elements
    - Process variables
    - Set points
    - Sensors
  - Control loop paradigm
Software paradigm for control systems con’t

- Results in a particular kind of dataflow architecture
  - In addition to providing data to each other the paradigm assumes that data is updated continuously
- Requires a cyclic topology
- Asymmetry between the control element and the process element
Solution 2: process-control view

- A control-view architecture might be appropriate when software is embedded involving continuous behaviour
- The cruise-control system is supposed to maintain constant speed in an automobile despite variations in terrain, load, air resistance, fuel quality, ...
Solution 2: process-control view con’t

• Identify the essential system elements
  • Computational elements
    • Process definition: the process receives a throttle setting and turns the wheels
      • The process takes a throttle setting as input and controls the speed of the vehicle
    • Control algorithm: the algorithm models the current speed from the wheel pulses, compares it to the desired speed and changes the throttle setting
      • Clock input needed
      • The current throttle setting must be maintained
Identify the essential system elements

- Data elements
  - Controlled variable: current speed of the vehicle
  - Manipulated variable: the throttle setting
  - Set point: desired speed, several inputs
  - Sensor for controlled variable: current speed
    - Modeled on data from wheel pulses using the clock
Solution 2: process-control view con’t

- Two subproblems:
  - Whenever the system is active determine the desired speed
  - Control the engine throttle setting to maintain the desired speed
    - This is the actual control problem
Solution 2: process-control view con’t

- Control architecture for the control system:
  - Model the current speed from the wheel pulses
    - Where should the wheel pulses be taken from?
    - Has the controller full control authority over the process?
Solution 2: process-control view con’t

- Set point computation:
  - Two inputs representing dataflows
    - Active/inactive
    - Desired speed
  - The controller is a continuously evaluating function that matches the dataflow character of the inputs and outputs
  - Two parts:
    - Is the system active?
    - Determine the desired speed
Solution 2: process-control view
con’t

• Summary
  • The objects in the OO view have roles in the resulting system
    • Use the control-loop architecture for the system as a whole
    • Other architectures to elaborate the elements
Analysis and discussion

- The selection of an architecture commits the designer to a particular view of the problem
- OO architectures are supported by methodologies
- Methodologies for control-loops?
A methodology should help designer to decide when the architecture is appropriate.

A methodology should help the designer to identify the elements of the design and their interactions:
- Find the objects in oo.

A methodology should help the designer to identify critical decisions:
- Safety problems in control.