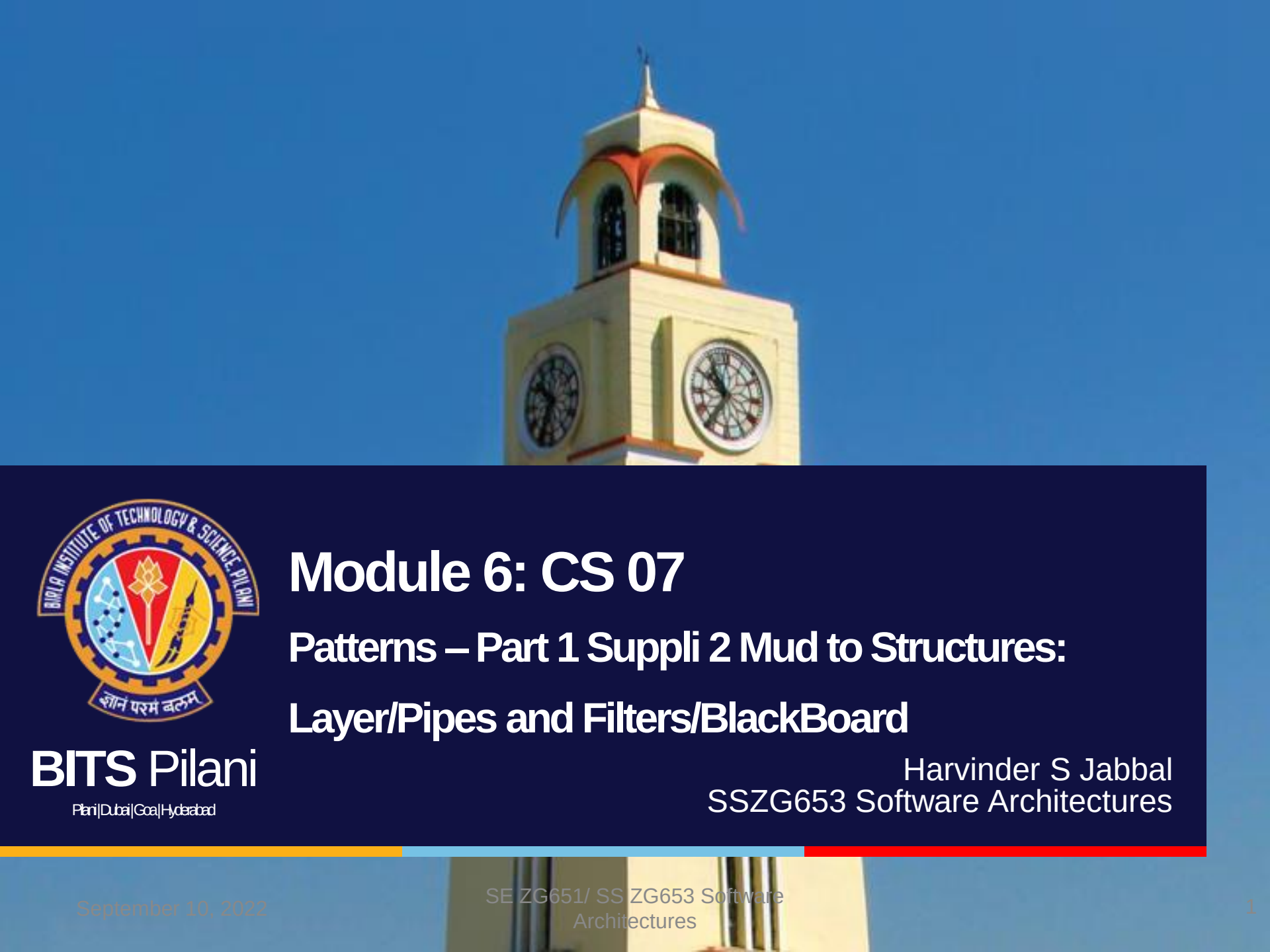


Module 6: CS 07

Patterns – Part 1 Suppli 2 Mud to Structures: Layer/Pipes and Filters/BlackBoard

Harvinder S Jabbal
SSZG653 Software Architectures

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
  - the Layers pattern
  - the Pipes and Filters pattern
  - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
  - Broker
- and refers to two patterns in other categories,
  - Microkernel
  - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
  - the Model-View-Controller pattern (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

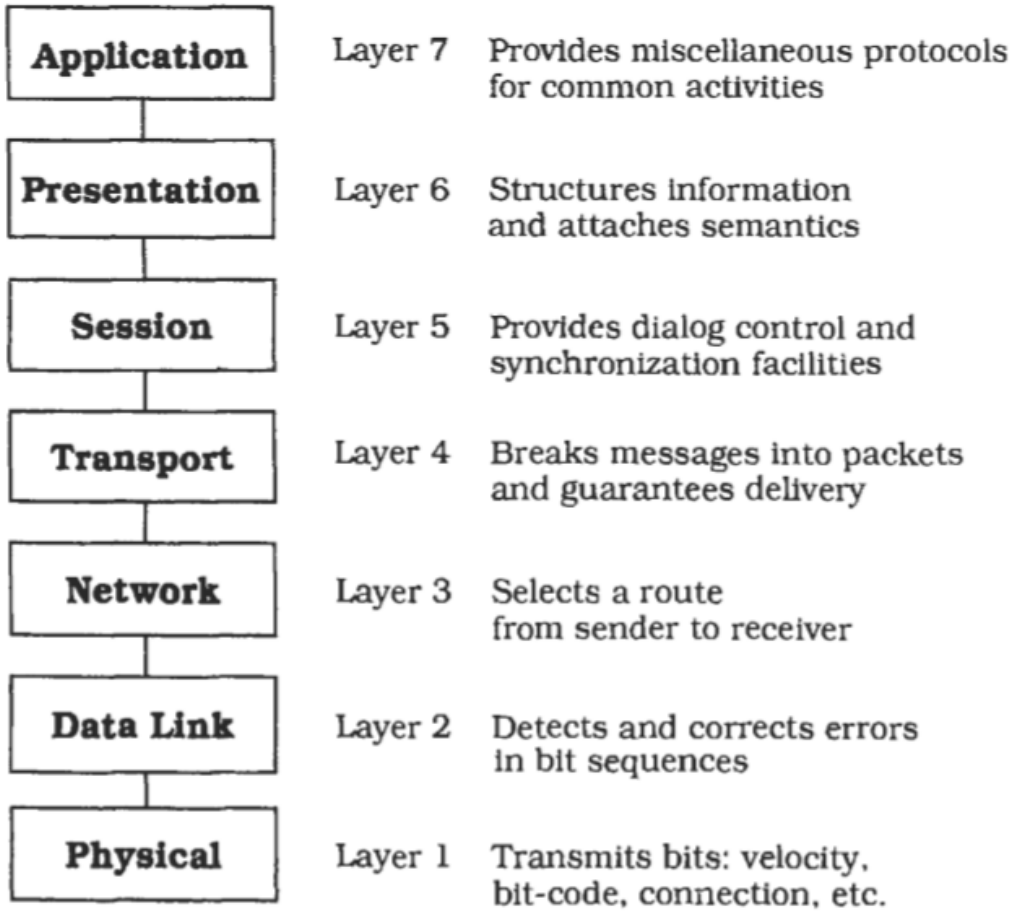
- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

From Mud to Structure

- the Layers pattern
- the Pipes and Filters pattern
- the Blackboard pattern

From Mud to Structure: Layers

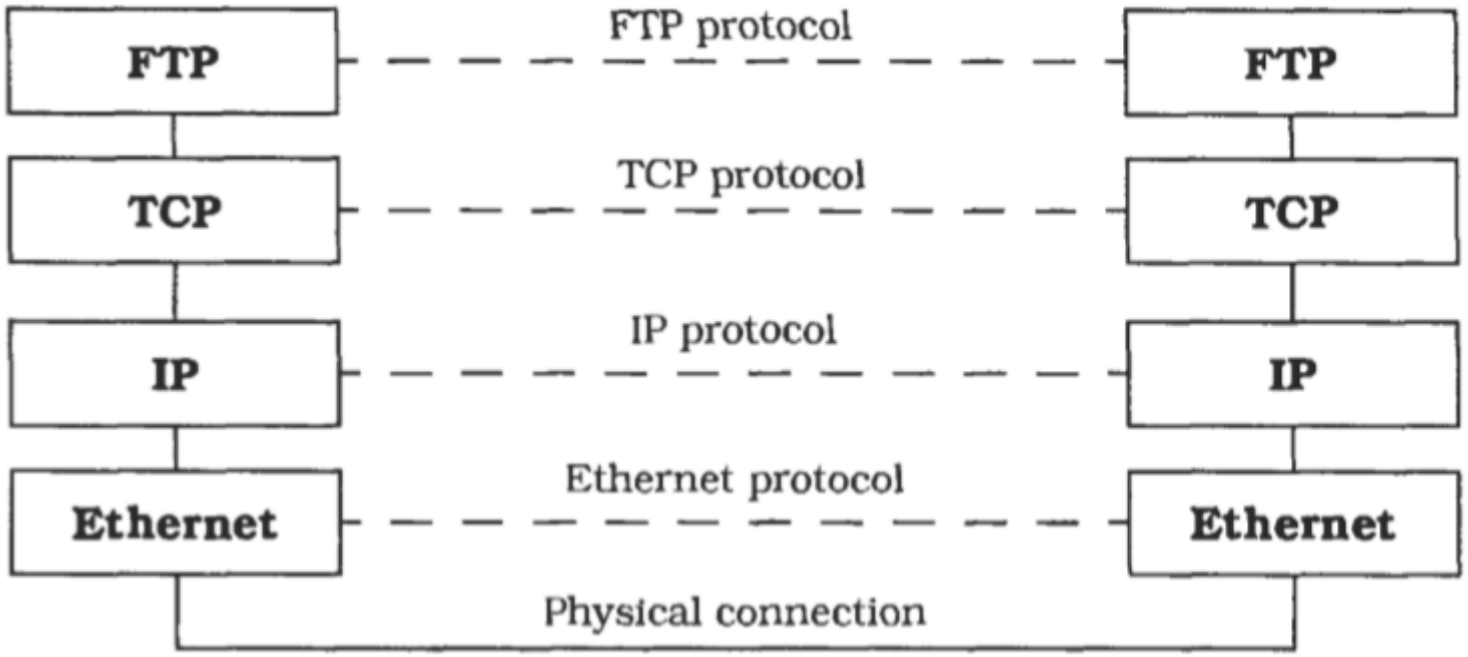
Layers- OSI 7 Layer Model



Implementation Steps

1. Define the abstraction criterion for grouping tasks into layers.
2. Determine the number of abstraction levels according to your abstraction criterion.
3. Name the layers and assign tasks to each of them.
4. Specify the services.
5. Refine the layering.
6. Specify an interface for each layer.
7. Structure individual layers.
8. Specify the communication between adjacent layers.
9. Decouple adjacent layers.
10. Design an error-handling strategy.

Variant of OSI: TCP/IP



Variant: Relaxed Layered System

- This is a variant of the Layers pattern that is less restrictive about the relationship between layers.
- In a Relaxed Layered System each layer may use the services of all layers below it, not only of the next lower layer.
- A layer may also be partially opaque- this means that some of its services are only visible to the next higher layer, while others are visible to all higher layers.
- The gain of flexibility and performance in a Relaxed Layered System is paid for by a loss of maintainability.
- This is often a high price to pay, and you should con- sider carefully before giving in to the demands of developers asking for shortcuts.
- We see these shortcuts more often in infrastructure systems, such as the UNIX operating system or the X Window System, than in application software.
- The main reason for this is that infra-structure systems are modified less often than application systems, and their performance is usually more important than their maintainability.

Variant: Layering Through Inheritance.

innovate achieve lead

- This variant can be found in some object-oriented systems.
- In this variant lower layers are implemented as base classes.
- A higher layer requesting services from a lower layer inherits from the lower layer's implementation and hence can issue requests to the base class services.
- An advantage of this scheme is that higher layers can modify lower-layer services according to their needs.
- A drawback is that such an inheritance relationship closely ties the higher layer to the lower layer.
- If for example the data layout of a C++ base class changes, all subclasses must be recompiled.
- Such unintentional dependencies introduced by inheritance are also known as the fragile base class problem.

Known Usages:

Virtual Machines. We can speak of lower levels as a virtual machine that insulates higher levels from low-level details or varying hardware. For example, the Java Virtual Machine (JVM) defines a binary code format. Code written in the Java programming language is translated into a platform-neutral binary code, also called byte- codes, and delivered to the JVM for interpretation. The JVM itself is platform-specific-there are implementations of the JVM for different operating systems and processors. Such a two-step translation process allows platform-neutral source code and the delivery of binary code not readable to humans1, while maintaining platform- independency.

Known Usages:

APIs. An Application Programming Interface is a layer that encapsu- lates lower layers of frequently-used functionality. An API is usually a flat collection of function specifications, such as the UNIX system calls. 'Flat' means here that the system calls for accessing the UNIX file system. These libraries provide the benefit of portabil- ity between different operating systems, and provide additional higher-level services such as output buffering or formatted output. They often carry the liability of lower efficiency2, and perhaps more tightly-prescribed behavior, whereas conventional system calls would give more flexibility-and more opportunities for errors and concep- tual mismatches, mostly due to the wide gap between high-level application abstractions and low-level system calls.

Known Usages:

Information Systems (IS) from the business software domain often use a two-layer architecture. The bottom layer is a database that holds company-specific data. Many applications work concurrently on top of this database to fulfill different tasks. Mainframe interactive systems and the much-extolled Client-Server systems often employ this architecture. Because the tight coupling of user interface and data representation causes its share of problems, a third layer is introduced between them-the domain layer-which models the conceptual structure of the problem domain. As the top level still mixes user interface and application, this level is also split, resulting in a four-layer architecture. These are, from highest to lowest:

Presentation

Application logic

Domain layer

Database

Known Usages:

Windows NT [Cus93]. This operating system is structured according to the Microkernel pattern (171). The NT Executive component corresponds to the microkernel component of the Microkernel pattern. The NT Executive is a Relaxed Layered System, as described in the Variants section. It has the following layers:

System services: the interface layer between the subsystems and the NT Executive.

2. Input/output buffering in Mgher layers is often intended to have the inverse effect-better performance than undisciplined direct use of lower-level system calls.

@ Resource management layer: this contains the modules Object Manager, Security Reference Monitor, Process Manager, 1/0 Manager, Virtual Memory Manager and Local Procedure Calls.

@ Kernel: this takes care of basic functions such as interrupt and ex- ception handling, multiprocessor synchronization, thread schedu- ling and thread dispatching.

@ HAL (Hardware Abstraction Layer): this hides hardware differences between machines of different processor families.

@ Hardware

Windows NT relaxes the principles of the Layers pattern because the Kernel and the 1/0 manager access the underlying hardware directly for reasons of efficiency.

onsequences

Benefits

Benefits

1. Reuse of layers.
2. Support for standardization.
3. Dependencies are kept local

Liabilities

1. Cascades of changing behaviour.
2. Lower efficiency.
3. Unnecessary work.
4. Difiulty of establishing the correct granularity of layers.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Pipes and Filters

Pipes and Filters

The Pipes and Filters architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters. Recombining filters allows you to build families of related systems.

Pipe and Filter Pattern

innovate achieve lead

Context: Many systems are required to transform streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts.

Problem: Such systems need to be divided into reusable, loosely coupled components with simple, generic interaction mechanisms. In this way they can be flexibly combined with each other. The components, being generic and loosely coupled, are easily reused. The components, being independent, can execute in parallel.

Solution: The pattern of interaction in the pipe-and-filter pattern is characterized by successive transformations of streams of data. Data arrives at a filter's input port(s), is transformed, and then is passed via its output port(s) through a pipe to the next filter. A single filter can consume data from, or produce data to, one or more ports.

Problem

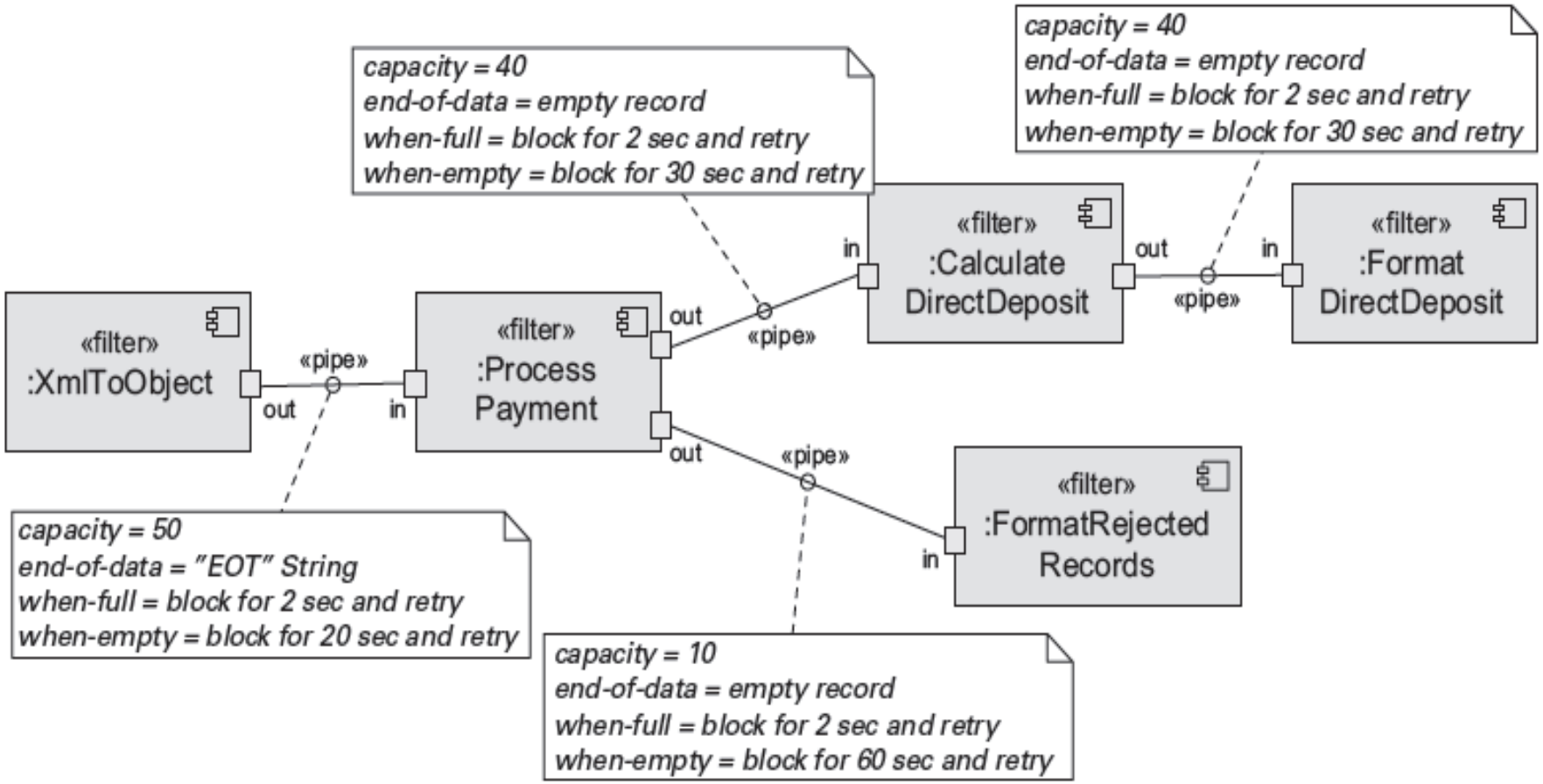
- Imagine you are building a system that must process or transform a stream of input data. Implementing such a system as a single component may not be feasible for several reasons:
  - the system has to be built by several developers,
  - the global system task decomposes naturally into several processing stages, and
  - the requirements are likely to change.
- You therefore plan for future flexibility by exchanging or reordering the processing steps.
- By incorporating such flexibility, it is possible to build a family of systems using existing processing components.

Problem

The design of the system-especially the interconnection of processing steps-has to consider the following forces:

- Future system enhancements should be possible by exchanging processing steps or by recombination of steps, even by users.
- Small processing steps are easier to reuse in different contexts than large components. Non-adjacent processing steps do not share information.
- Different sources of input data exist, such as a network connection or a hardware sensor providing temperature readings, for example. It should be possible to present or store final results in various ways. Explicit storage of intermediate results for further processing in files clutters directories and is error-prone, if done by users.
- You may not want to rule out multi-processing the steps, for example running them in parallel or quasi-parallel.

Pipe and Filter Example



Solution

- The Pipes and Filters architectural pattern divides the task of a system into several sequential processing steps.
- These steps are connected by the data flow through the system-the output data of a step is the input to the subsequent step.
- Each processing step is implemented by a filter component.
- A filter consumes and delivers data incrementally-in contrast to consuming all its input before producing any output-to achieve low latency and enable real parallel processing.
- The input to the system is provided by a data source such as a text file.
- The output flows into a data sink such as a file, terminal, animation program and so on.
- The data source, the filters and the data sink are connected sequentially by pipes.
- Each pipe implements the data flow between adjacent processing steps.
- The sequence of filters combined by pipes is called a processing pipeline.

Pipe and Filter Solution

Overview: Data is transformed from a system's external inputs to its external outputs through a series of transformations performed by its filters connected by pipes.

Elements:

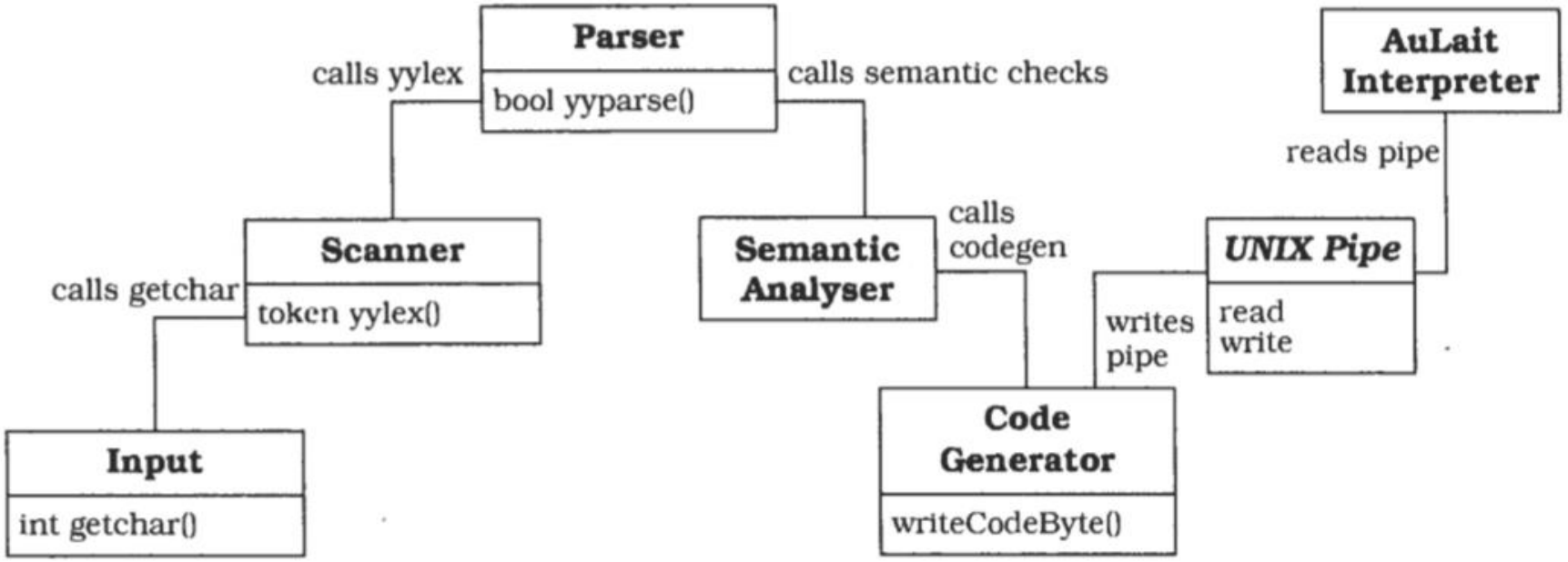
- *Filter,* which is a component that transforms data read on its input port(s) to data written on its output port(s).
- *Pipe,* which is a connector that conveys data from a filter's output port(s) to another filter's input port(s). A pipe has a single source for its input and a single target for its output. A pipe preserves the sequence of data items, and it does not alter the data passing through.

Relations: The *attachment* relation associates the output of filters with the input of pipes and vice versa.

Constraints:

- Pipes connect filter output ports to filter input ports.
- Connected filters must agree on the type of data being passed along the connecting pipe.

Another Example



Implementation Steps

1. Divide the system's task into a sequence of processing stages.
2. Define the data format to be passed along each pipe.
3. Decide how to implement each pipe connection.
4. Design and implement the fiters.
5. Design the error handling.
6. Set up the processing pipeline.
   - If your system handles a single task you can use a standardized main program that sets up the pipeline and starts processing. This type of system may benefit from a direct- call pipeline, in which the main program calls the active filter to start processing.
   - You can increase flexibility by providing a shell or other end-user facility to set up various pipelines from your set of filter components.
   - Such a shell can support the incremental development of pipelines by allowing intermediate results to be stored in files, and supporting files as pipeline input.
   - You are not restricted to a text-only shell such as those provided by UNIX, and could even develop a graphical environment for visual creation of pipelines using 'drag and drop' interaction.

Variant: Tee and join pipeline systems.

innovate achieve lead

- The single-input slngle-output filter specification of the Pipes and Filters pattern can be varied to allow filters wlth more than one input and/or more than one output.
- Processing can then be set up as a directed graph that can even contain feedback loops.
- The design of such a system, especially one with feedback loops, requires a solid foundation to explain and understand the complete calculation- a rigorous theoretical analysis and specification using formal methods are appropriate, to prove that the system terminates and produces the desired result.

Known Usage: UNIX

- UNIX popularized the Pipes and Filters paradigm.
- The command shells and the availability of many filter programs made this approach to system development popular.
- As a system for software developers, frequent tasks such as program compilation and documentation creation are done by pipelines on a 'traditional' UNIX system.
- The flexibility of UNIX pipes made the operating system a suitable platform for the binary reuse of filter programs and for application integration.

Known Usage: CMS Pipelines

innovate achieve lead

- CMS Pipelines is an extension to the operating system of IBM mainframes to support Pipes and Filters architectures.
- The implementation of CMS pipelines follows the conventions of CMS, and defines a record as the basic data type that can be passed along pipes, instead of a byte or ASCII character.
- CMS Pipelines provides a reuse and integration platform in the same way as UNIX.
- Because the CMS operating system does not use a uniform I/O-model in the same way as UNIX, CMS Pipelines defines device drivers that act as data sources or sinks, allowing the handling of specific I/O-devices within pipelines.

Known Usage: LASSP Tools

- LASSP Tools is a toolset to support numerical analysis and graphics.
- The toolset consists mainly of filter programs that can be combined using UNIX pipes.
- It contains graphical input devices for analog input of numerical data using knobs or sliders, filters for numerical analysis and data extraction, and data sinks that produce animations from numerical data streams.

Benefits

1. No intermediate files necessary, but possible.
2. Flexibility by filter exchange.
3. Flexibility by recombination.
4. Reuse of filter components.
5. Rapid prototyping of pipelines.
6. Egiciemy by parallel processing.

Liabilities

1. Sharing state information is expensive or inflxible.
2. Efficiency gain by parallel processing is often an illusion.
3. Data transformation overhead.
4. Error handling.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Black Board

Blackboard

- The Blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known.
- In Blackboard several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution.
- **CONTEXT**
- An immature domain in which no closed approach to a solution is known or feasible.

Problem

- The Blackboard pattern tackles problems that do not have a feasible deterministic solution for the transformation of raw data into high-level data structures, such as diagrams, tables or English phrases.
- Vision, image recognition, speech recognition and surveillance are examples of domains in which such problems occur.
- They are characterized by a problem that, when decomposed into sub-problems, spans several fields of expertise.
- The solutions to the partial problems require different representations and paradigms.
- In many cases no predetermined strategy exists for how the 'partial problem solvers' should combine their knowledge.
- This is in contrast to functional de-composition, in which several solution steps are arranged so that the sequence of their activation is hard-coded.

forces that influence solutions to these problems

- A complete search of the solution space is not feasible in a reasonable time.
- Since the domain is immature, you may need to experiment with different algorithms for the same subtask. Individual modules should be easily exchangeable.
- There are different algorithms that solve partial problems. Unrelated logics, representations, algorithms, paradigms or domains may be involved.
- An algorithm usually works on the results of other algorithms.
- Uncertain data and approximate solutions are involved.
- Employing disjoint algorithms induces potential parallelism. If possible you should avoid a strictly sequential solution.

Solution

- The idea behind the Blackboard architecture is a collection of independent programs that work cooperatively on a common data structure.
- Each program is specialized for solving a particular part of the overall task, and all programs work together on the solution.
- These specialized programs are independent of each other.
- They do not call each other, nor is there a predetermined sequence for their activation.
- Instead, the direction taken by the system is mainly determined by the current state of progress.
- A central control component evaluates the current state of processing and coordinates the specialized programs.
- This data-directed control regime is referred to as opportunistic problem solving.
- It makes experimentation with different algorithms possible, and allows experimentally-derived heuristics to control processing.
- During the problem-solving process the system works with partial solutions that are combined, changed or rejected.
- Each of these solutions represents a partial problem and a certain stage of its solution.
- The set of all possible solutions is called the solution space, and is organized into levels of abstraction.
- The lowest level of solution consists of an internal representation of the input.
- Potential solutions of the overall system task are on the highest level.
- The name 'blackboard was chosen because it is reminiscent of the situation in which human experts sit in front of a real blackboard and work together to solve a problem.
- Each expert separately evaluates the current state of the solution, and may go up to the blackboard at any time and add, change or delete information.
- Humans usually decide themselves who has the next access to the blackboard.
- In the pattern we describe, a moderator component decides the order in which programs execute if more than one can make a contribution.

Structure

- Divide your system into a component called blackboard, a collection of knowledge sources, and a control component.
- The blackboard is the central data store. Elements of the solution space and control data are stored here.
- We use the term vocabulary for the set of all data elements that can appear on the blackboard.
- The blackboard provides an interface that enables all knowledge sources to read from and write to it.
- All elements of the solution space can appear on the blackboard.
- For solutions that are constructed during the problem solving process and put on the blackboard, we use the terms hypothesis or blackboard entry.
- Hypotheses rejected later in the process are removed from the blackboard.

Components

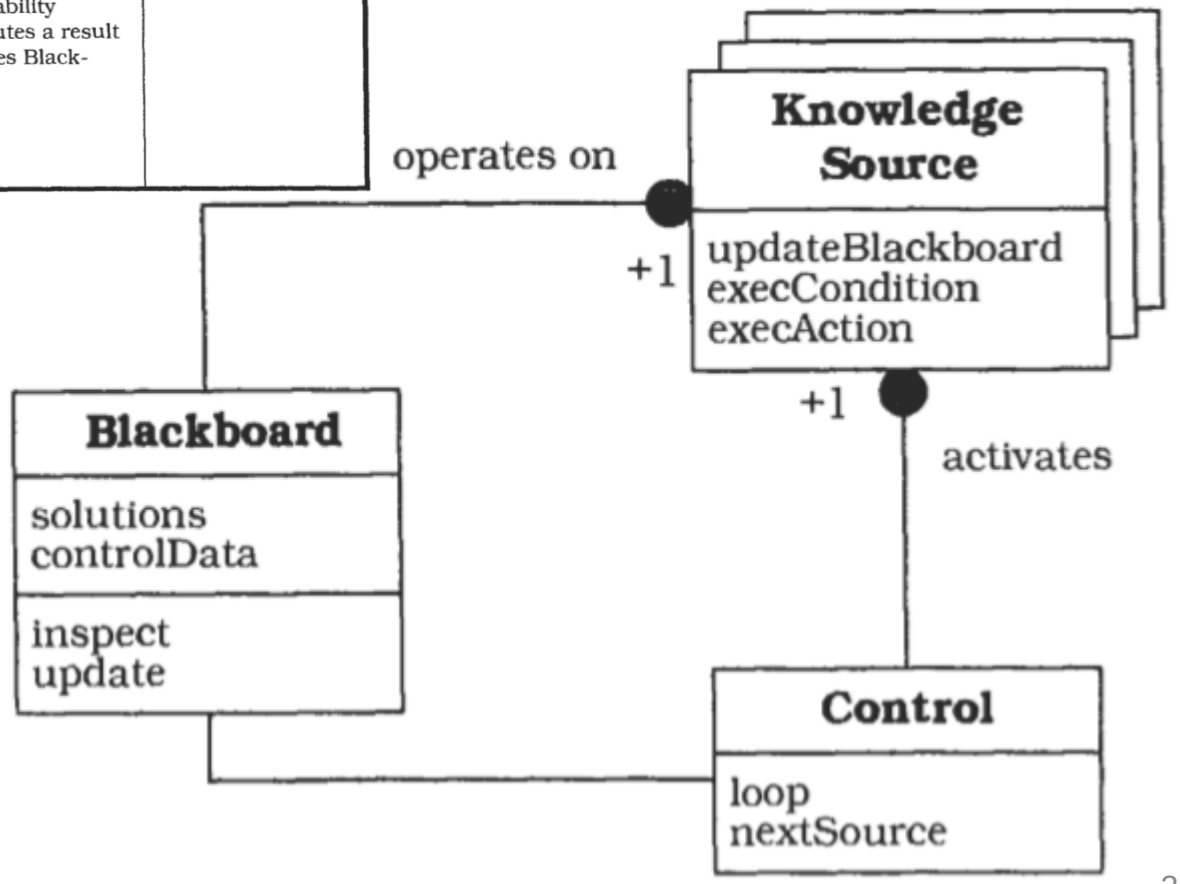
- **The blackboard** can be viewed as a three-dimensional problem space with the time line for speech on the X-axis, increasing levels of abstraction on the Y-axis and alternative solutions on the Z-axis
- **Knowledge sources** are separate, independent subsystems that solve specific aspects of the overall problem. Together they model the overall problem domain. None of them can solve the task of the system alone-a solution can only be built by integrating the results of several knowledge sources.
- The **control component** runs a loop that monitors the changes on the blackboard and decides what action to take next. It schedules knowledge source evaluations and activations according to a knowledge application strategy. The basis for this strategy is the data on the blackboard.

Structure

Class Blackboard	Collaborators -
Responsibility • Manages central data	

Class Knowledge Source	Collaborator • Blackboard
Responsibility • Evaluates its own applicability • Computes a result • Updates Blackboard	

Class Control	Collaborators • Blackboard • Knowledge Source
Responsibility • Monitors Blackboard • Schedules Knowledge Source activations	



Steps in implementing Blackboard

1. Define the problem.
2. Define the solution spacefor the problem.
3. Divide the solution process into steps.
4. Divide the knowledge into specialized knowledge sources with certain subtasks.
5. Define the vocabulary of the blackboard.
6. Specify the control ofthe system.

Variant: Production System

- This architecture is used in the OPS language.
- In this variant subroutines are represented as condition-action rules, and data is globally available in working memory.
- Condition-action rules consist of a left-hand side that specifies a condition, and a right-hand side that specifies an action.
- The action is executed only if the condition is satisfied and the rule is selected.
- The selection is made by a 'conflict resolution module’.
- A Blackboard system can be regarded as a radical extension of the original production system formalism: arbitrary programs are allowed for both sides of the rules, and the internal complexity of the working memory is increased.
- Complicated scheduling algorithms are used for conflict-resolution.

Variant: Repository.

- This variant is a generalization of the Blackboard pattern.
- The central data structure of this variant is called a repository.
- In a Blackboard architecture the current state of the central data structure, in conjunction with the Control component, finally activates knowledge sources.
- In contrast, the Repository pattern does not specify an internal control.
- A repository architecture may be controlled by user input or by an external program.
- A traditional database, for example, can be considered as a repository.
- Application programs working on the database correspond to the knowledge sources in the Blackboard architecture

Known Usages

HEARSAY-11.

- The first Blackboard system was the HEARSAY-I1 speech recognition system from the early 1970's. It was developed as a natural language interface to a literature database.

HASP/SIAP.

- The HASP system was designed to detect enemy submarines. In this system, hydrophone arrays monitor a sea area by collecting sonar signals.

CRYSALIS.

- This system was designed to infer the three-dimensional structure of protein molecules from X-ray diffraction data

TRICERO.

- This system monitors aircraft activities

SUS: 'Software Understanding System

- In a matching process the system compares patterns from a pattern base to the system under analysis.
- SUS incrementally builds a 'pattern map' of the analyzed software that then can be viewed.

Concusion

Thank You

Credits:

Text Books