Software Architecture

Gerson Sunyé
University of Nantes
gerson.sunye@univ-nantes.fr
http://sunye.free.fr

Agenda

- Introduction
- Development view
- Physical view
- Process view
- Reliability view
- Architectural styles

Introduction

Software Architecture

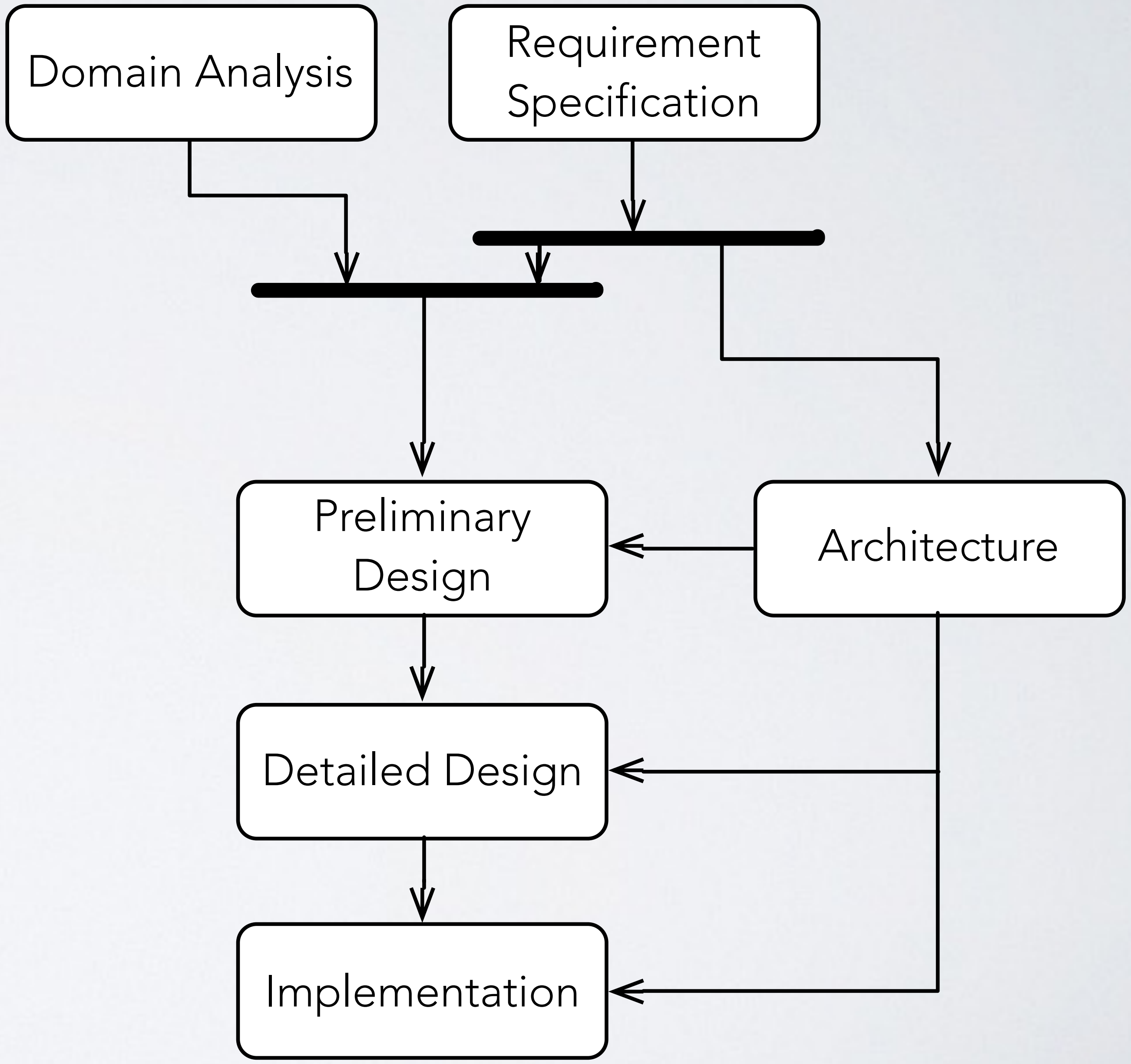
Software architecture is the high level structure of a software system, the discipline of creating such structures, and the documentation of these structures. It is the set of structures needed to reason about the software system, and comprises the software elements, the relations between them, and the properties of both elements and relations. [Clements et al. 2010].

A set of Decisions and Constraints

- Strategical choices for system design and implementation.
- Some examples (with different granularities):
  - Use a n-tiers architecture.
  - Use REST for remote procedure call.
- Use the MVC framework for partitioning business classes and user interfaces.
- Respect the Google Java Style for Java source code.
- Access methods should start with get and set.
- Use a graph database for persisting data.

An Orthogonal Activity

- Architectural design may be independent from the domain analysis.
- The same architecture may be used on different projects.



Architectural Views

- Logical view: describes architecturally significant elements of the architecture and the relationships between them.
- Process view: describes the concurrency and communications elements of an architecture.
- Physical view: depicts how the major processes and components are mapped on to the applications hardware.
- Development view: captures the internal organization of the software components as held in e.g. a configuration management tool.
- Architecture use cases: capture the requirements for the architecture; related to more than one particular view

[Kruchten 1995]

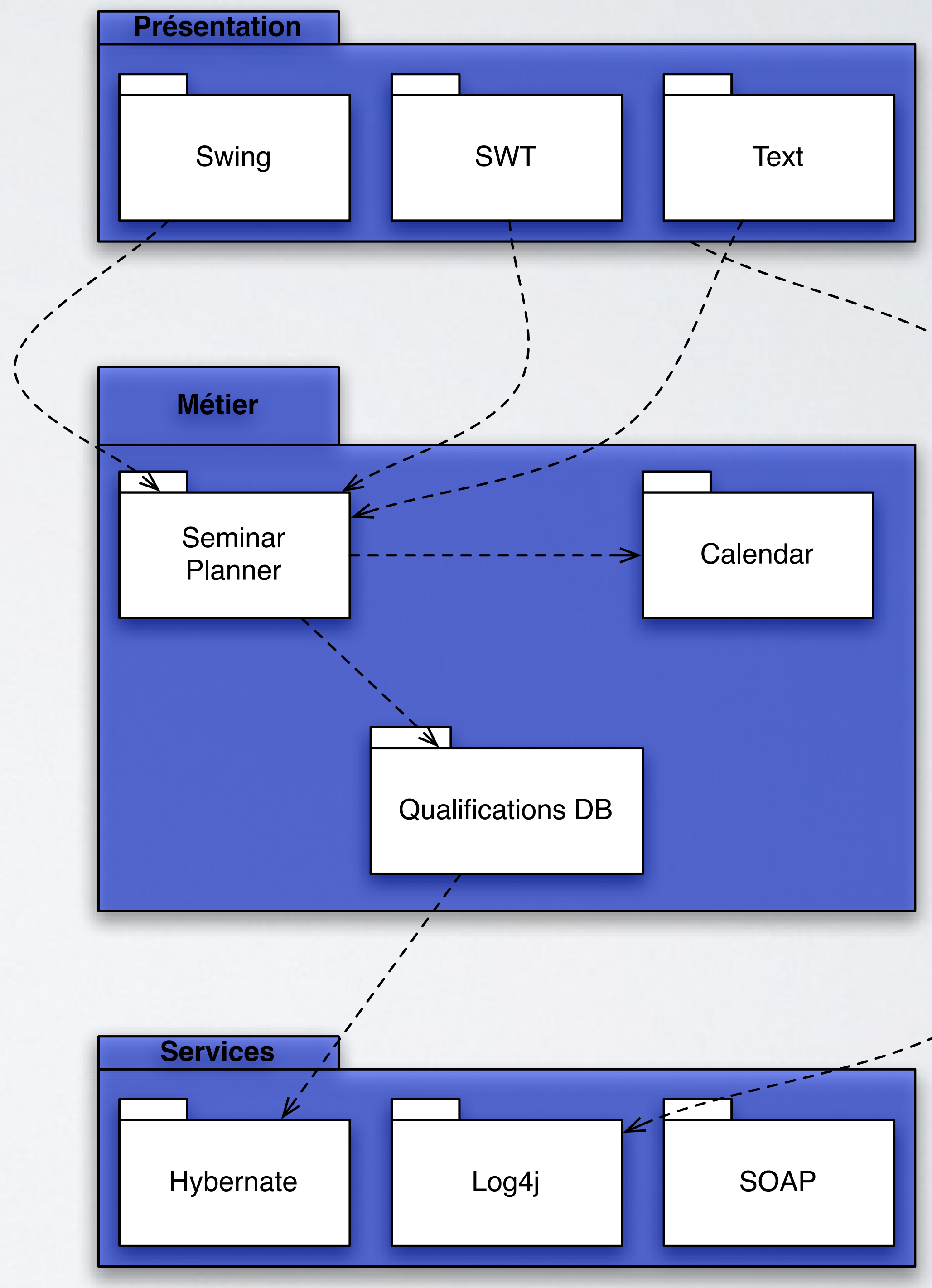
Development View

Development View

- Describes how programmers should organize the software source code:
  - Dependencies, versioning, code generation, build, etc.

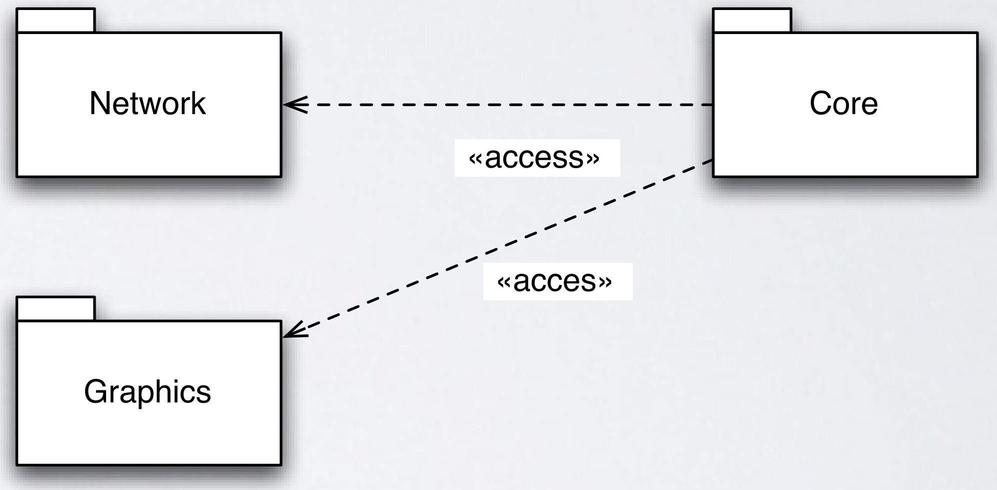
Development View

- UML Package diagram
- Dependencies between packages.



Dependencies between Packages

- UML Dependency relations:
  - «import»: elements are imported from one namespace to another.
  - «access»: elements from the target namespace are simply used.
  - «merge»: defines how a package extends another.



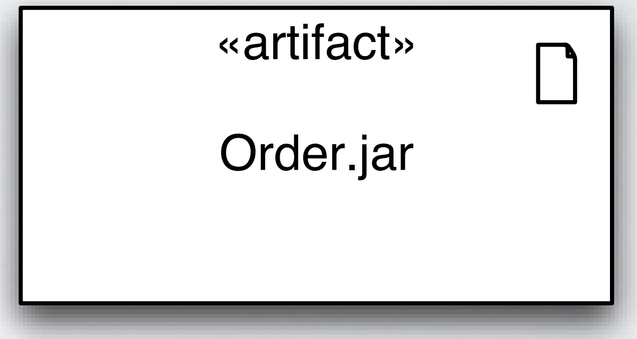
Physical View

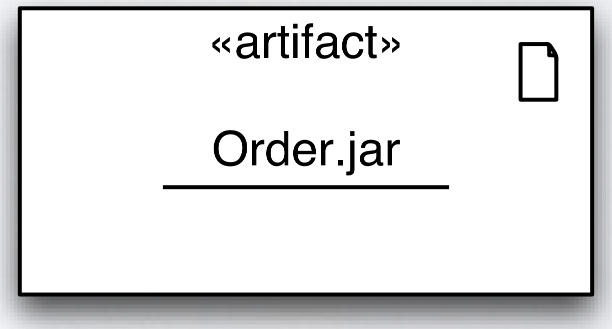
Physical View

- Describes how system engineers should deploy software artifacts on logical nodes.
- Defines the network topology (node organization) and communication paths.
- Specify deployment constraints for artifacts: required hardware, libraries, resources, etc.).

Artifacts

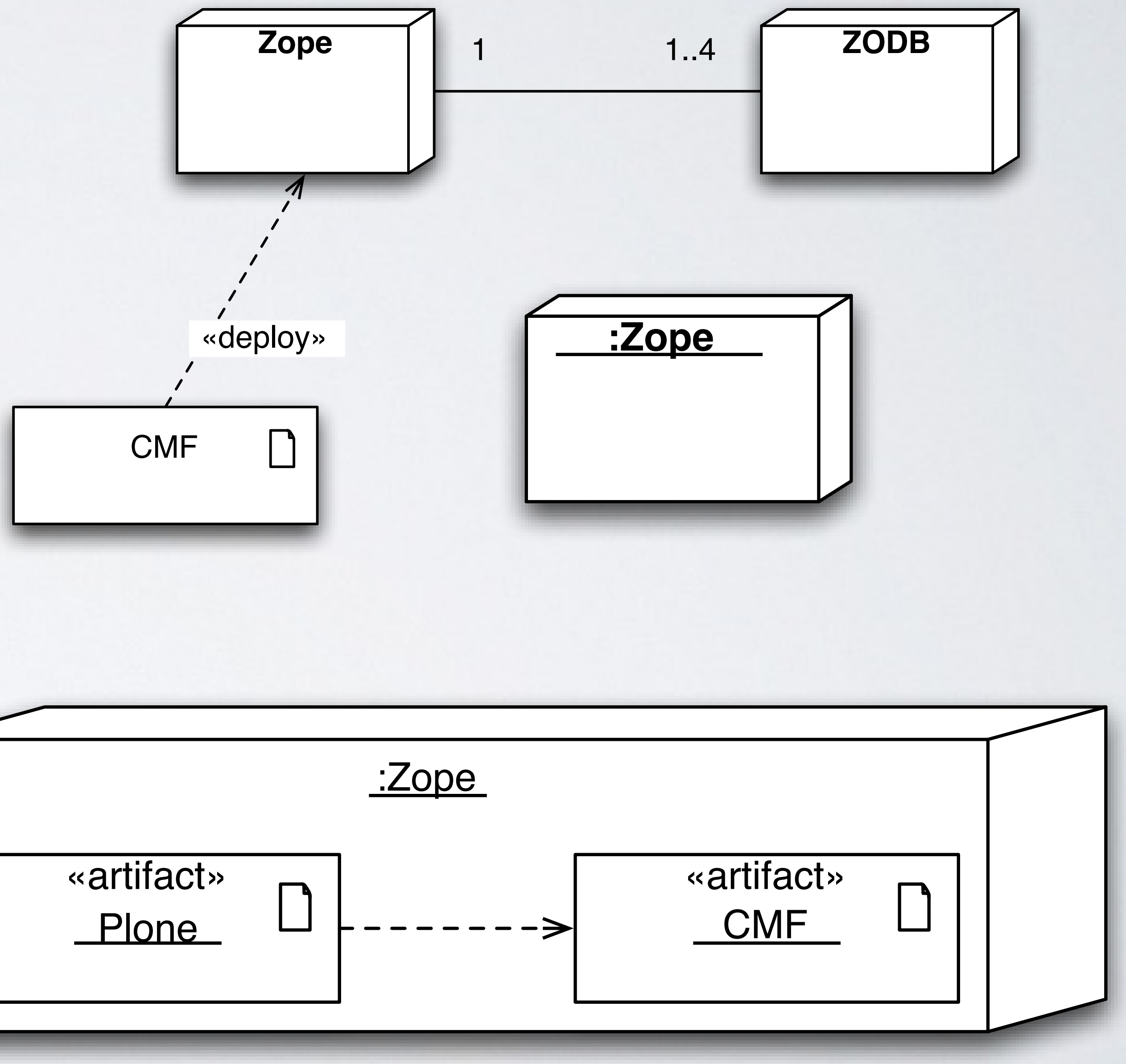
- Artifacts are physical pieces, used during the software development process:
  - source files, models, scripts, binaries, documents, configuration files, etc.





Nodes

- Nodes are logical resources, on which artifacts are deployed.
- Nodes are connected through communication paths.
- Node instances may also be represented.



Process View

Process View

- Describes the process flow of control mode and process concurrency for each executable unit (processes, threads).
- If each component executes in its own process, detailed information should be provided.
- Related quality factors: performance and scalability.

Flow of Control

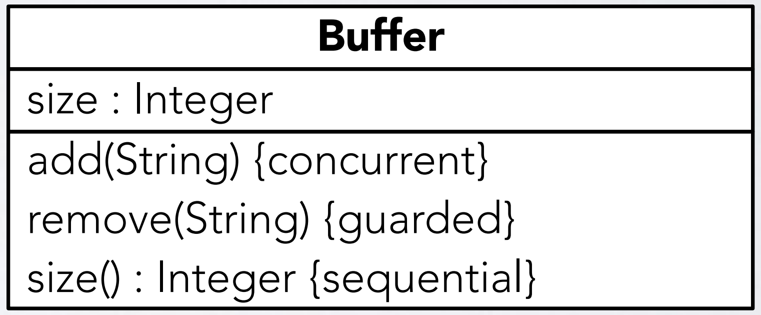
- Sequential: there is a single flow of control. i.e, one thing, and one thing only, can take place at a time.

- Concurrent: there are multiple simultaneous flow of control i.e, more than one thing can take place at a time.

- Other modes: procedural, event-driven, batch, etc.

Synchronization

- Synchronization means arranging the flow of controls of objects so that mutual exclusion is guaranteed.
- Approaches to handle synchronization:
    1. Sequential: clients must coordinate outside the object so that only one flow is in the object at a time
    2. Guarded: multiple flow of control is sequentialized with the help of object's guarded operations. in effect it becomes sequential.
    3. Concurrent: multiple flow of control is guaranteed by treating each operation as atomic

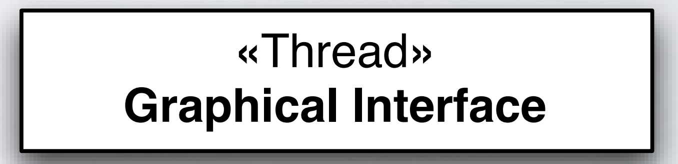


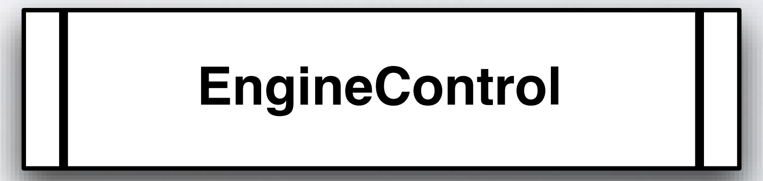
Concurrency

- May be intrinsic, imposed by the environment: multiples user interfaces, etc. For instance:
  - Swing creates a separate thread for the user interface.
  - Java RMI creates a thread for remote procedure calls.
- Or a design choice:
  - Parallel processing
  - Redundancy

Active Classes

- Active classes are just classes which represents an independent flow of control
- When an active object is created, the associated flow of control is started; when the active object is destroyed, the associated flow of control is terminated
- Standard stereotypes that apply to active classes are,
    1. «process»: specifies a heavyweight flow that can execute concurrently with other processes.
    2. «thread» – Specifies a lightweight flow that can execute concurrently with other threads within the same process.





Active Class Implementation

- Naive approach:
  - Assign a process/thread for each active objet.
- Consequences:
  - Context Switching Overhead
  - Wasted system resources: inactive processes waiting for new events.

Other Approaches

- Concurrency design patterns:
  - Proactor.
  - Reactor.
  - Thread pool
- Actors (Scala, Erlang).

Concurrent Data Structures

- Atomic objets: Integer, String, etc.
- Priority queue.
- Blocking queue.
- Concurrent associative array.

Concurrency patterns [Douglass02]

- Concurrency
- Interrupt
- Guarded Call
- Rendezvous
- Cyclic Executive
- Round Robin
- Static Priority
- Dynamic Priority

Logical View

Logical View

- Use a component diagram to show components and their provided and required interfaces.
- Describe component responsibilities
- Other important information, e.g.,
  - Dependencies
  - Non-functional properties (e.g., maximum memory usage, reliability, required test coverage, etc.)
  - Mandated implementation technology (e.g., EJB, COM, etc.)

Reliability View

Reliability View

- Redundancy specification
- Limit conditions forecast:
  - Initializations
  - Finalization
- Failure management and restart.

Failure Management

- Typically:
  - Retry operation
  - Use redundant information to correct failure
  - Go to a fail-safe state
  - Alert someone
  - Restart the system

Reliability Patterns [Douglass02]

- Watchdog: a simple polling process.
- Protected Single Channel
- Homogeneous Redundancy
- Triple Modular Redundancy
- Heterogeneous Redundancy
- Monitor-Actuator
- Sanity Check
- Safety Executive

Architectural Styles

Architectural Style

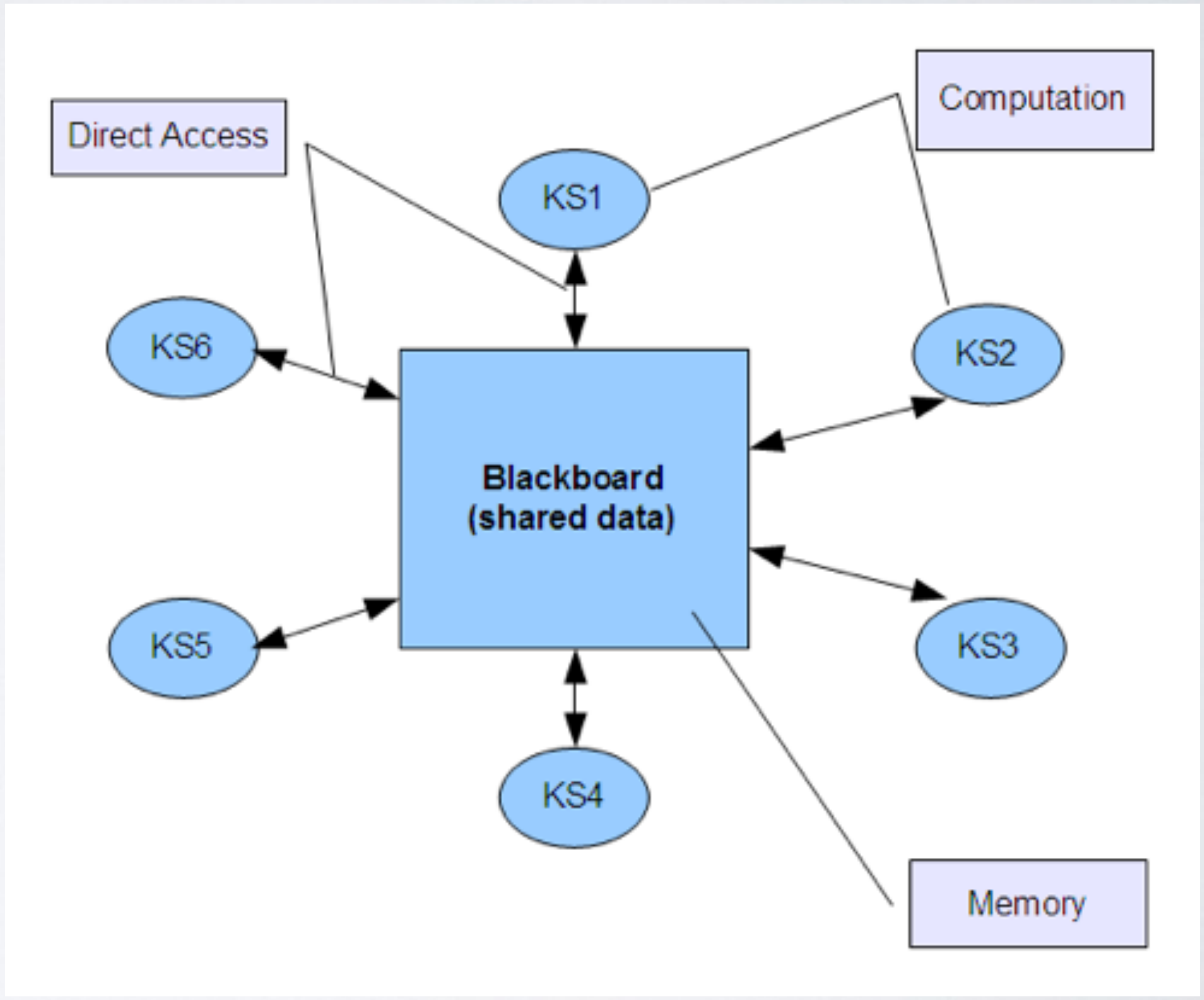
- “A set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem, together with local or global constraints on the way the composition is done”. [Shaw and Clements, 1996]

Most Known Architectural Styles

- Blackboard
- Client-server (2-tier, 3-tier, n-tier, cloud computing exhibit this style)
- Component-based
- Data-centric
- Event-driven (or Implicit invocation)
- Layered
- Monolithic application
- Peer-to-peer (P2P)
- Pipes and filters
- Plug-ins
- Representational state transfer (REST)
- Rule-based
- Service-oriented
- Shared nothing architecture
- Space-based architecture
- Persistence Free architecture

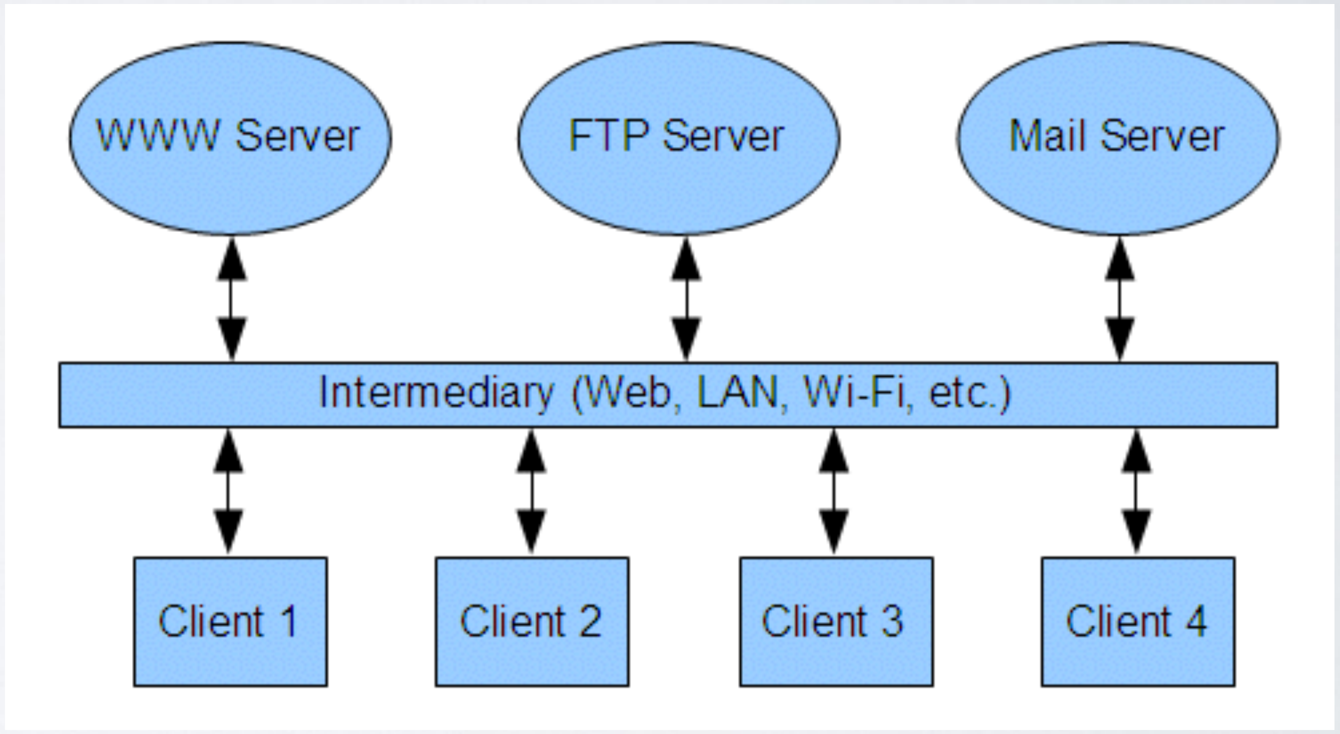
Blackboard Style

- All shared data is held in a central database that can be accessed by all subsystems.



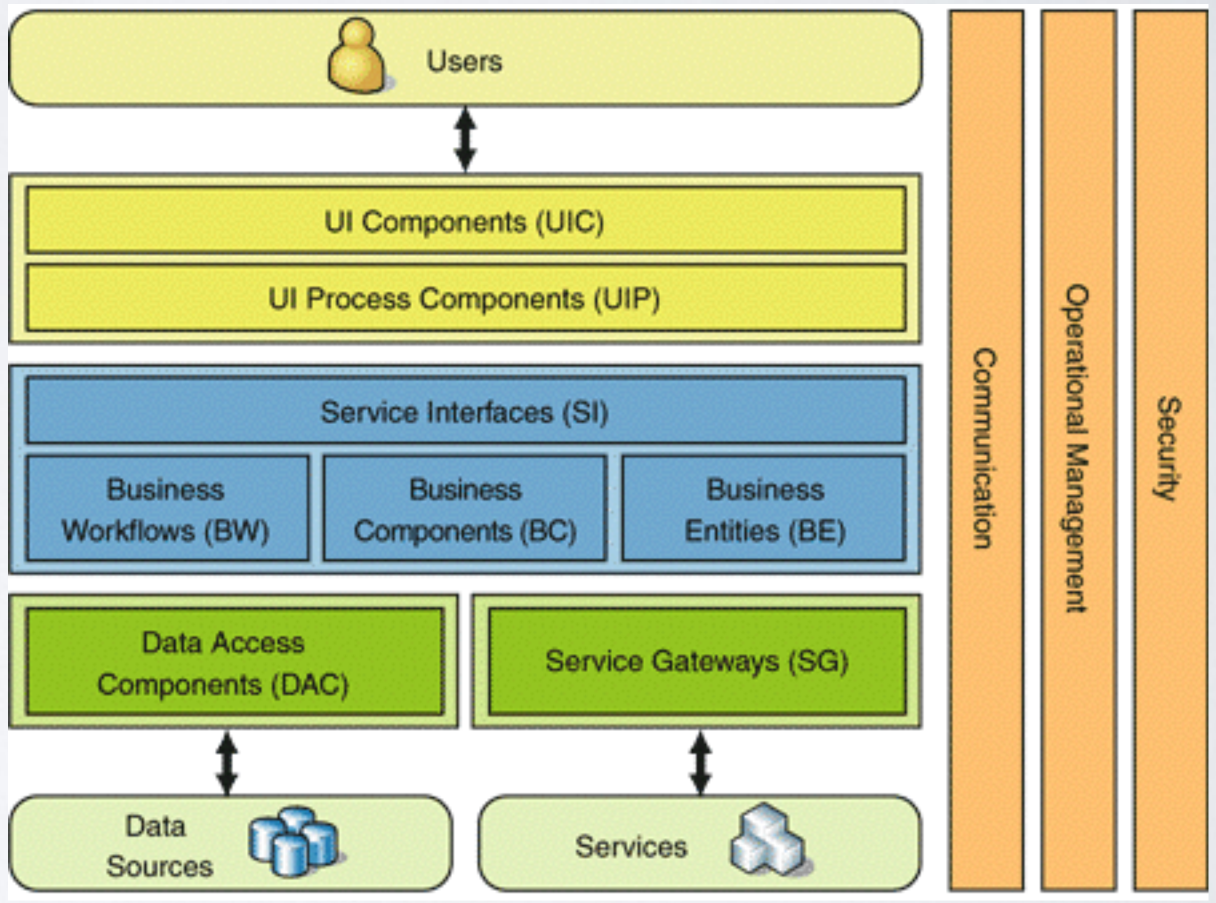
Client-Server Style

- The system is organized as a set of services and their associated servers and clients that access and use those services.



Layered Style

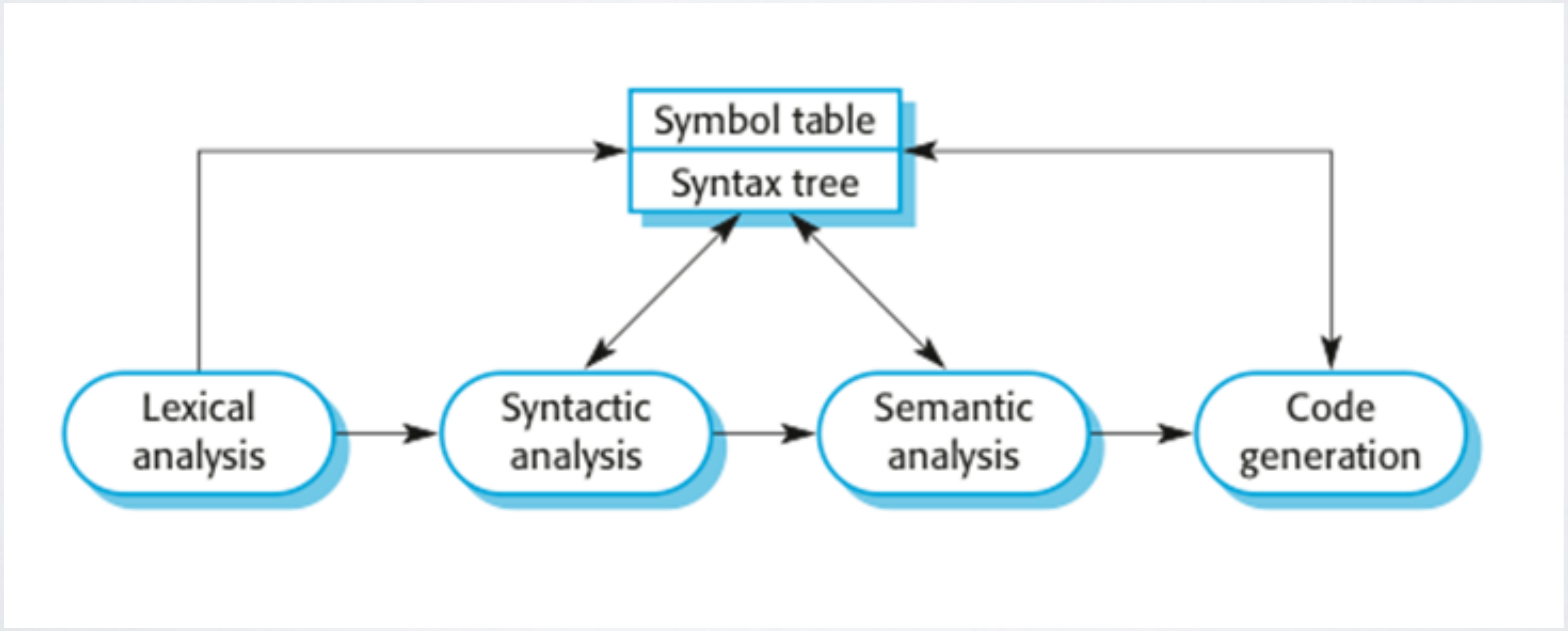
- Hierarchical organization:
- Each layer:
  - provides service to the layer above it and
  - serves as a client to the layer below it.



[https://msdn.microsoft.com/en-us/library/ms978678.aspx]

Pipes and Filters Style

- Function-Oriented Pipelining
- A sequence of filters that transform (filter) data before passing it on via pipes to other filters.



Conclusion

Further Readings

- Architectural Patterns
  - Layers
  - Pipes and Filters
  - Broker
- Reliability Patterns
  - Watchdog
- Design Patterns
  - Proxy
  - Observer
- Concurrency Patterns
  - Reactor
  - Message Queueing

References

- “Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns”. Bruce Powel Douglass. Addison-Wesley Professional, 1999.
- “Design Patterns: Elements of Reusable Object-Oriented Software”. Erich Gamma,Richard Helm,Ralph Johnson, John Vlissides. Pearson Education, 31 oct. 1994.
- “Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects”. Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. John Wiley & Sons, 22 avr. 2013
- “Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems”. Bruce Powel Douglass. Broché, 23 septembre 2002.

References

- "Documenting Software Architectures: Views and Beyond", Second Edition. Clements, Paul; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford. Boston: Addison-Wesley., 2010. ISBN 0-321-55268-7.

- "Applied Software Architecture". Christine Hofmeister, Robert Nord, Dilip Soni. Addison-Wesley, 1999.

- "SA Styles, Patterns, and Tactics". Henry Muccini. DISIM, University of L'Aquila. (Advanced Software Engineering Course).

Software Architecture