Module 3: Architecture

In the sense of information systems

- Web architectures
- Enterprise architectures
- Interoperation architectures
- Message-oriented middleware

Architecture Conceptually

- How a system is organized
- An over-used, vaguely defined term
  - Software architecture
  - Standards, e.g., Berners-Lee’s “layer cake”
  - May include processes
  - May include human organizations
Understanding Architecture

- Two main ingredients of a system
  - Components
  - Interconnections

- Openness entails specifying the interconnections cleanly
  - Physical components disappear
  - Their logical traces remain

- Information environments mean that the interconnections are protocols

Understanding Protocols

- Protocols encapsulate interactions
  - Connect: conceptual interfaces
  - Separate: provide clean partitions among logical components

- Wherever we can identify protocols, we can
  - Make interactions explicit
  - Enhance reuse
  - Improve productivity
  - Identify new markets and technologies

- Protocols yield standards; their implementations yield products
Architectural Examples

When viewed architecturally, each logical component class serves some important function

- Power: UPS
- Network connectivity
- Storage: integrity, persistence, recovery
- Policy management
- Decision-making
- Knowledge and its management

What are some products in the above component classes?

IT Architectures

The term is used more broadly in IT settings

- The organization of an IT system
- The extensibility and modifiability of a system
- Even the governance of a system
IT Governance

The human management of IT systems

- The human organization in a system taken broadly
- Even the processes by which a system is updated or upgraded (including the human aspects such as permissions)
- Nontechnical aspects, such as flows of responsibility

Used to be confused with architecture, but now increasingly separated

Enterprise Models: 1

- Capture static and dynamic aspects of enterprises
- Document information resources
  - Databases and knowledge bases
  - Applications, business processes, and the information they create, maintain, and use
Enterprise Models: 2

- Capture organizational structure
- Document business functions
  - Rationales behind designs of databases and knowledge bases
  - Justifications for applications and business processes

Enterprise Models: 3

By being explicit representations, models enable

- Integrity validation
- Reusability
- Change impact analysis
- Automatic database and application generation via CASE tools
Enterprise Architecture Objectives

At the top-level, to support the business objectives of the enterprise; these translate into

- Accommodating change by introducing new
  - Applications
  - Users
  - Interfaces and devices
- Managing information resources
  - Preserving prior investments, e.g., in legacy systems
  - Upgrading resources
- Developing blueprints to guide resource and application installation and decommissioning

Enterprise Architecture Observations

Continual squeeze on funds, staffing, and time available for IT resources

- Demand for rapid development and deployment of applications
- Demand for greater ROI
- Essential tension
  - Need to empower users and suborganizations to ensure satisfaction of their local and of organizational needs
  - Ad hoc approaches with each user or each suborganization doing its own IT cause failure of interoperability
Enterprise Architecture Principles

Business processes should drive the technical architecture

- Define dependencies and relationships among users and suborganizations of an organization
- Message-driven approaches are desirable because they decouple system components
- Event-driven approaches are desirable because they help make a system responsive to events that are potentially visible and significant to users

Architecture Modules: Applications

Often most visible to users

- Application deployment
- Data modeling and integrity
- Business intelligence: decision support and analytics
- Interoperation and cooperation
  - Ontologies: representations of domain knowledge
- Component and model repositories
- Business process management
Architecture Modules: Systems

Functionality used by multiple applications
- Middleware: enabling interoperation, e.g., via messaging
- Identity management
- Security and audit
- Accessibility
- Policy repositories and engines

Architecture Modules: Infrastructure

- Connectivity
- Platform: hardware and operating systems
- Storage
- System management
Enterprise Functionalities: 1

It helps to separate the key classes of functionality in a working software system

- Presentation: user interaction
  - A large variety of concerns about device constraints and usage scenarios
- Business logic
  - Application logic
  - General rules

Enterprise Functionalities: 2

- Data management
  - Ensuring integrity, e.g., entity and referential integrity (richer than storage-level integrity)
  - Enabling access under various kinds of problems, e.g., network partitions
  - Supporting recovery, e.g., application, operating system, or hardware failures
Enterprise Functionalities: 3

Bases for choosing the above three-way partitioning as opposed to some other

- Size of implementations
- Organizational structure: who owns what and who needs what
- Staff skill sets
  - User Interface: usability and design
  - Programming
  - Database
  - Policy tools
- Products available in the marketplace

One-Tier and Two-Tier Architectures

- One tier: monolithic systems; intertwined in the code base
  - Historically the first
  - Common in legacy systems
  - Difficult to maintain and scale up
- Two-tier: separate data from presentation and business logic
  - Classical client-server (or fat client) approaches
  - Mix presentation with business rules
  - Change management
Three-Tier Architecture: 1

- **Presentation tier or frontend**
  - Provides a view to user and takes inputs
  - Invokes the same business logic regardless of interface modalities: voice, Web, small screen, ...

- **Business logic tier or middle tier**
  - Specifies application logic
  - Specifies business rules
    - Application-level policies
    - Inspectable
    - Modifiable

Three-Tier Architecture: 2

- **Data tier or backend**
  - Stores and provides access to data
  - Protects integrity of data via concurrency control and recovery
Multitier Architecture

Also known as n-tier (sometimes treated synonymously with three-tier)

- Best understood as a componentized version of three-tier architecture where
  - Functionality is assembled from parts, which may themselves be assembled
  - Supports greater reuse and enables greater dynamism
  - But only if the semantics is characterized properly
- Famous subclass: service-oriented architecture

Architectural Tiers Evaluated

The tiers reflect logical, not physical partitioning

- The more open the architecture the greater the decoupling among components
  - Improves development through reuse
  - Enables composition of components
  - Facilitates management of resources, including scaling up
  - Sets boundaries for organizational control
- In a narrow sense, having more moving parts can complicate management
- But improved architecture facilitates management through divide and conquer
How About Database Triggers?

- **Pros**: essential for achieving high efficiency
  - Reduce network load and materializing and serializing costs
  - Leave the heavy logic in the database, under the care of the DBA

- **Cons**: rarely port well across vendors
  - Difficult to introduce and manage because of DBA control
  - Business rules are context-sensitive and cannot always be applied regardless of how the data is modified
Implementational Architecture: 1

Centered on a Web server that
- Supports HTTP operations
- Usually multithreaded

Implementational Architecture: 2

Application server
- Mediates interactions between browsers and backend databases: runs computations, invoking DB transactions as needed
- Provides a venue for the business logic
- Different approaches (CGI, server scripts, servlets, Enterprise JavaBeans)
Implementational Architecture: 3

Database Servers

- Hold the data, ensuring its integrity
- Manage transactions, providing
  - Concurrency control
  - Recovery

Transaction monitors can manage transactions across database systems, but within the same administrative domain

Data Center Architecture

- Demilitarized zone (DMZ)
  - External router
  - Load balancer
- Firewall: only the router can contact the internal network
  - Internal network
  - Web servers
  - Application servers
  - Database servers
Application Servers

Architectural abstraction separating business logic from infrastructure

- Load balancing
- Distribution and clustering
- Availability
- Logging and auditing
- Connection (and resource) pooling
- Security

Separate programming from administration roles

Middleware: 1

Components with routine, reusable functionality

- Abstracted from the application logic or the backend systems
- Any functionality that is being repeated is a candidate for being factored out into middleware
- Enables plugging in endpoints (e.g., clients and servers) according to the stated protocols
- Often preloaded on an application server
- Simplify programmer’s task and enable refinements and optimizations
Middleware: 2

Software components that implement architectural interfaces, e.g., transaction, persistence, . . .

- **Explicit:**
  - Invoke specialized APIs explicitly
  - Difficult to create, maintain, port

- **Implicit:**
  - Container invokes the appropriate APIs
  - Based on declarative specifications
  - Relies on request interceptions or reflection

Containers

- Discussed above in connection with EJBs
- Architectural abstraction geared for hosting business components
  - Remote method invocation
  - Threading
  - Messaging
  - Transactions
Message-Oriented Middleware: 1

- **Queues**: point to point, support posting and reading messages
- **Topics**: logical multicasts, support publishing and subscribing to application-specific topics; thus more flexible than queues
- Can offer reliability guarantees of delivery or failure notification to sender
  - Analogous to store and forward networks
- Some messages correspond to event notifications

Message-Oriented Middleware: 2

- Varies in reliability guarantees
- Usually implemented over databases
- Can be used through an invocation-based interface (i.e., registered callbacks)
Message-Driven Beans

A standardized receiver for messages

- Clients can’t invoke them directly; must send messages to them
- No need for specialized interfaces, such as home, remote, . . .
- Easy interface to implement: mainly `onMessage()`, but limited message typing
- Stateless: thus no conversations

Methods for Message-Driven Beans

- `onMessage()`: define what actions to take when a message arrives on the destination this bean is watching
Peer-to-Peer Computing

- **Symmetric client-server:** (callbacks) each party can be the client of the other
- **Asynchrony:** while the request-response paradigm corresponds to pull, asynchronous communication corresponds to push
  - Generally to place the entire intelligence on the server (pushing) side
- **Federation of equals:** (business partners) when the participants can enact the protocols they like

Web Architecture

Principles and constraints that characterize Web-based information systems

- **URI:** Uniform Resource Identifier
- **HTTP:** HyperText Transfer Protocol
- Metadata must be recognized and respected
  - Enables making resources comprehensible across administrative domains
  - Difficult to enforce unless the metadata is itself suitably formalized
Uniform Resource Identifier: 1

- URIs are abstract
- What matters is their (purported) uniqueness
- URIs have no proper syntax per se
- Kinds of URIs include
  - URLs, as in browsing: not used in standards any more
  - URNs, which leave the mapping of names to locations up in the air

Uniform Resource Identifier: 2

Good design requirements

- Ensure that the identified resource can be located
- Ensure uniqueness: eliminate the possibility of conflicts through appropriate organizational and technical means
- Prevent ambiguity
- Use an established URI scheme where possible
HTTP: HyperText Transfer Protocol

Intended meanings are quite strict, though not constrained by implementations

- Text-based, stateless
- Key verbs
  - Get
  - Post
  - Put
- Error messages for specific situations, such as resources not available, redirected, permanently moved, and so on

ReST: Representational State Transfer

Representational State Transfer

ReST is an architectural style for networked systems that constrains the connectors

- Models the Web as a network of hyperlinked resources, each identified by a URI
- Models a Web application as a (virtual) state machine
- A client selecting a link effects a state transition, resulting in receiving the next page (next state) of the application
Characteristics of ReST

- Client-Server
- Statelessness: in terms of sessions
  - What is an advantage of statelessness?
  - Where is the session state kept then?
- Focus on resources being manipulated and their representations being transferred
- Uniform Interface: URIs, hypermedia
- Caching: responses can be labeled as cacheable

Basic Interaction Models

Interactions among autonomous and heterogeneous parties

- Adapters: what are exposed by each party to enable interoperation
  - Sensors $\leftrightarrow$ information
  - Effectors $\Rightarrow$ actions
- Invocation-based adapters
- Message-oriented middleware
- Peer-to-peer computing
Invocation-Based Adapters: 1

Distributed objects (EJB, DCOM, CORBA)
- **Synchronous**: blocking method invocation
- **Asynchronous**: nonblocking (one-way) method invocation with callbacks
- **Deferred synchronous**: (in CORBA) sender proceeds independently of the receiver, but only up to a point

Invocation-Based Adapters: 2

Execution is best effort: application must detect any problems
- At most once
- More than once is
  - OK for idempotent operations
  - Not OK otherwise: application must check
DoDAF

Department of Defense Architecture Framework

- A standardized way to organize an enterprise architecture
- Lists 26 views organized into four categories
- Roughly, a software methodology
  - How to capture requirements: user activities
  - How to develop solutions: meet performance criteria
  - How to consider technical standards
- Best for large systems with lifetimes of decades