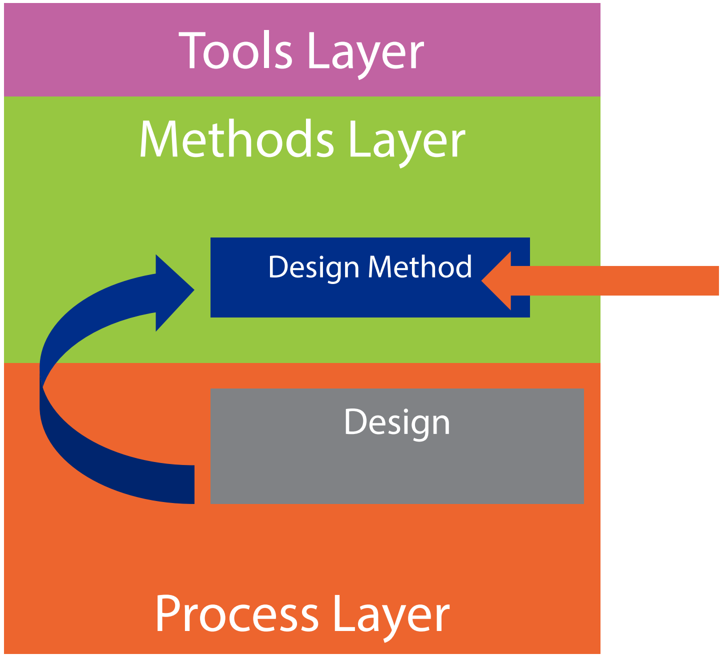
Design – Methodology

Mohamad Halabi
@mohamadhalabi

Introduction



Which Design Method to Use?

- **We studied two analysis methods:**
  - Structured Analysis
  - Object-Oriented Analysis
- **Both methods have corresponding design methods:**
  - Structured Design
  - Object-Oriented Design
- **Which method will we examine in this module?**

Structured Design

- **Structured Analysis criticism:**
  - Modeling is centered around data which does not properly model a system's building blocks
  - Behavioral modeling (STDs and CFDs) play a secondary role to data modeling (DFDs)
- **Structured Design is data-flow oriented**
  - Provides a transition from DFDs to architecture
  - However, only a Constructional viewpoint is covered
  - The viewpoint covers subsystem decomposition and runtime-invocation hierarchy
  - Main modeling notation is State Chart

Structured Design

- **Structured Design inherits Structured Analysis weaknesses**
  - Design is based on data-flow rather than objects
  - Only the Constructional viewpoint is covered
- **Structured Analysis And Design method could be appropriate for data-based systems**
- **It is not appropriate for real-time systems**
- **It is superseded by object-based methods**
  - Provide reusability
  - Better modeling of behavioral aspects
- **Therefore, Structured Design will not be covered**

Object-Oriented Design

- **Object-Oriented Analysis (OOA) is driven by objects to model the structure and behavior of a system**
- **Object-Oriented Design (OOD) follows Object-Oriented Analysis**
  - It uses analysis objects to drive system design
- **There are various approaches to Object-Oriented Design (ex: RUP)**
  - They all however share the same principles
- **I will adopt a generic OOD method, just as I did for OOA**
  - Based on the 4+1 View Model

Architecture-relevant Scenarios and Non-functional Requirements

- **In order to plan the architecture effort, an architect must:**
  - Identify architecture-relevant scenarios
  - Identify architecture-relevant non-functional requirements

Architecture-relevant Scenarios

- **Business-critical**
  - Ex: Customers searching for flights scenario
  - Scenario failure might result in loss of business
  - Architecture must take proper performance and availability measures
- **Business or technical risks**
  - Ex: Payment scenario integrates with 3rd party system
  - This system is a business risk
  - Architecture must tackle the unavailability of this system

Architecture-relevant Non-functional Requirements

- **Non-functional requirements can be quality attributes or constraints**
- **Quality attributes are mostly architecture-relevant**
  - Ex: Personalization is architecture-relevant for social media systems, but not so for an airline reservation system
  - Ex: Performance (ex: search operation latency) and security (ex: secure payment transactions) requirements are architecture-relevant
- **Constraints might or might not be architecture-relevant**
  - Ex: Governmental regulations could force certain quality measures
  - Ex: Technical constraints such as using a pre-existing middleware

Planning the Design Phase

- **After defining what is architecture-relevant, the architect can:**
  - Scope the work
  - Plan iterations
  - Plan activities within iterations

Plan Iterations

- **All evolutionary process models are based on iterative development**
- **Therefore, design most likely needs to be planned in iterations**
  - Design of viewpoints
  - Application of quality attributes
- **Therefore an architect must decide on number and scope of iterations**

Plan Activities

- Iteration activities will practically run in parallel
- For example design of a viewpoint and the application of a quality attribute, are two activities that run in parallel
- The architect must decide on dependencies between activities

Decide on Viewpoints

- **An architect must decide what viewpoints to use**
  - This depends on the nature of the system and stakeholder groups
- **We will use the 4 viewpoints of the 4+1 model**
  - Logical
  - Process
  - Physical
  - Development
- **In addition, two extra viewpoints will be used**
  - Information
  - Operation

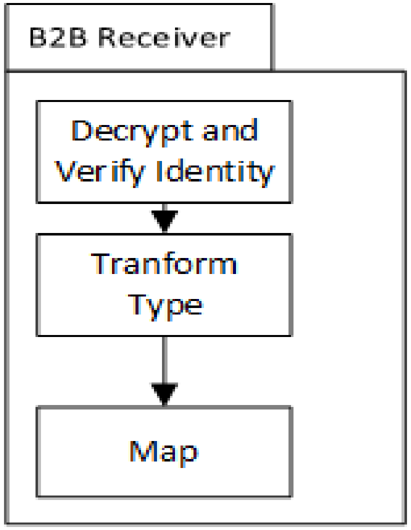
Decide on Notation

- **Opt for standard notation (such as UML) whenever possible**
- **Any notation can be used when standard notations are not sufficient**
  - Boxes
  - Circles
  - Arrows
  - Hand-made sketches
- **The important thing is that these notations are:**
  - Used consistently
  - Can be understood by the target stakeholder audience

Decomposition

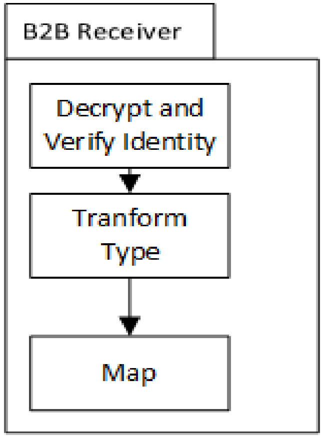
- **Decomposing a complex problem into smaller problems**
  - Simplifies finding solutions
  - Provides opportunities for reuse
- **The architecture must be decomposed to architecture-relevant components**
  - Structure, behavior, and quality attributes are studied in terms of these components
- **Q: To how many levels should architecture-relevant decomposition be applied?**
- **A: It's up to the architect's expert judgment**

Decomposition Example



- **Decision criteria:**
  1. Components are critical to B2B scenarios
     a) Failure in identification, decryption, or transformation might result in business loss
  2. Components affect multiple quality attributes
     a) “Decrypt” and Verify Identity” is critical to Security
     b) “Transform Type” and “Map” are critical to Performance

When Should We Stop Decomposing?



Should I decompose the three components?

1. **No, current level is enough.**
   - Define functional scope, dependencies, interfaces, messages, MEPs, application of quality attributes

2. **Yes, decompose into one more level of architecture-relevant components**
   - Also define all above design elements for the 3 components
   - But also decompose each of the three components into their constituent sub-components or classes
   - Apply point 1 design attributes on these constituents

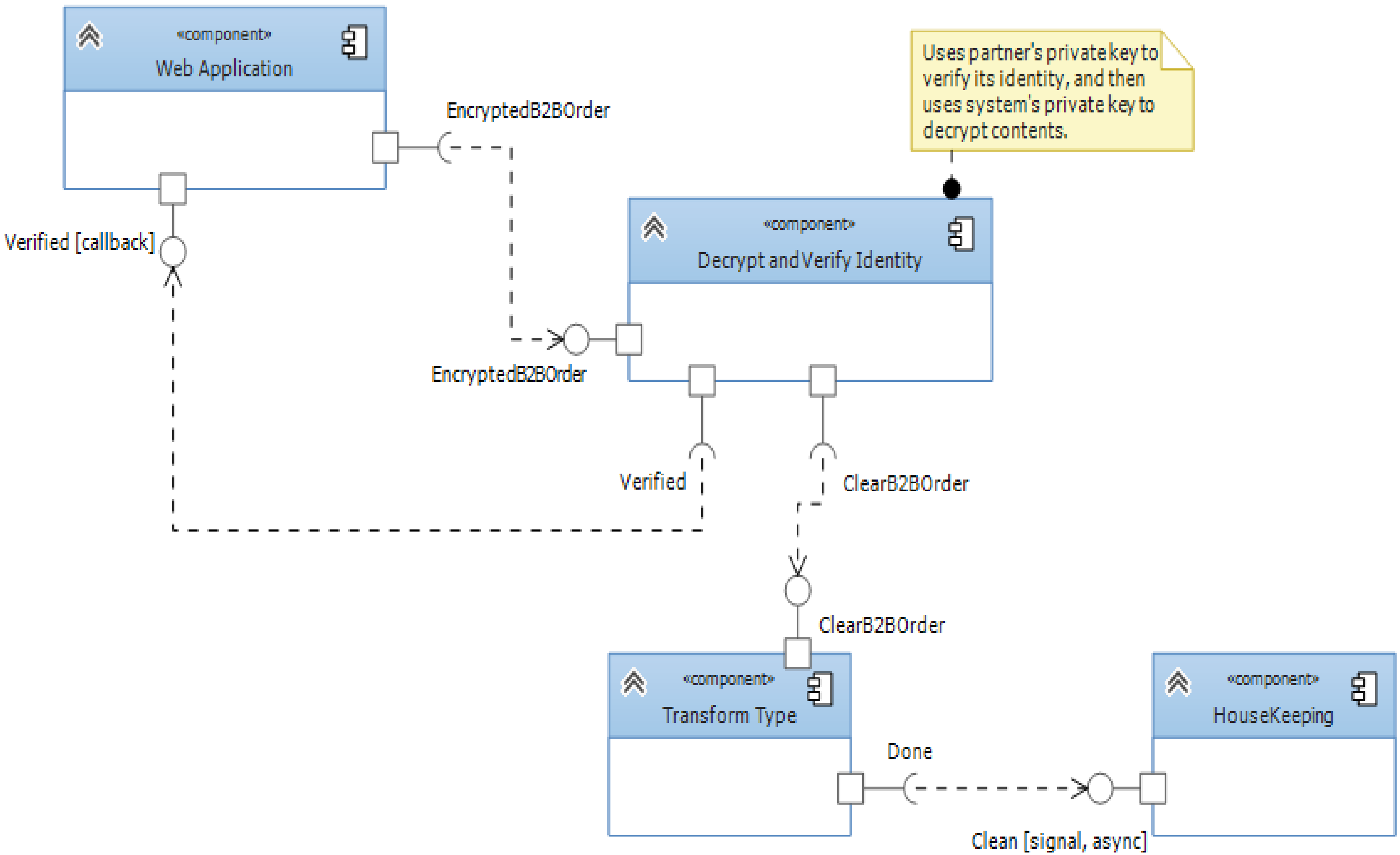
Identify Possible Architectural Styles

- **Styles are solutions to system-level organization problems**
  - They provide predefined components, responsibilities, and relationships
- **Styles provide the ability to reuse proven solutions:**
  - Cost (time and resources) reduction
  - Increased flexibility
  - Better understanding of quality attributes effect
  - Easier and more effective stakeholder communication
  - Better understanding of success measures
- **Ex: Pipe-and-Filter, Publish-and-Subscribe, Client/Server, N-Tier, Hub-and-Spoke, SOA, REST, etc…**

The Logical Viewpoint

- **Defines structure and behavior of the system**
  - Architecture-relevant components
  - Their relationships
  - Responsibilities (i.e. functional scope) of these components
  - Their interfaces
  - Their data and control flow interaction
    - Data flow: messages and MEPs (one-way, two-way, callbacks)
    - Control flow: how components signal each other to perform tasks

Example



«component»
Web Application
EncryptedB2BOrder
Uses partner's private key to verify its identity, and then uses system's private key to decrypt contents.
«component»
Decrypt and Verify Identity
Verified [callback]
EncryptedB2BOrder
Verified
ClearB2BOrder
ClearB2BOrder
«component»
Transform Type
«component»
HouseKeeping
Done
Clean [signal, async]

The Information Viewpoint

- **Defines the structure and behavior of architecture-relevant data**
  - Major data entities consumed and generated by architecture-relevant components
  - How are these entities structured and related
  - How information flows between components
    - Via interfaces defined in the Logical viewpoint
  - Data ownership
    - Systems or data stores that contain the up-to-date valid version of the data
  - Data security
  - Data creation
  - Data archiving
  - Data restoring
  - Data retirement
  - Data management

Notation

Structural Views

Class diagrams

- Data entities
- Relationships

Dynamic Views

Sequence diagrams

- Information flow between interfaces

State Machine diagrams

- State changes in response to events
- Data attribute changes as a result of state changes

- **Class, Sequence, and State Machine diagrams are covered in module 5**
- **They can be used at multiple abstraction levels**
  - Requirement modeling (i.e. model the problem domain)
  - Design modeling (i.e. model the solution domain)
- **Diagrams and text can be used to model concerns such as ownership, archiving, and storage**

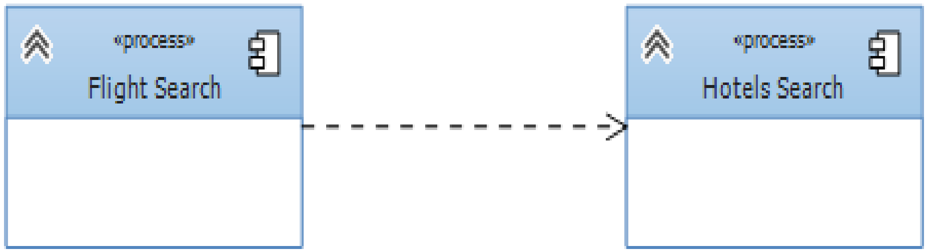
The Process Viewpoint

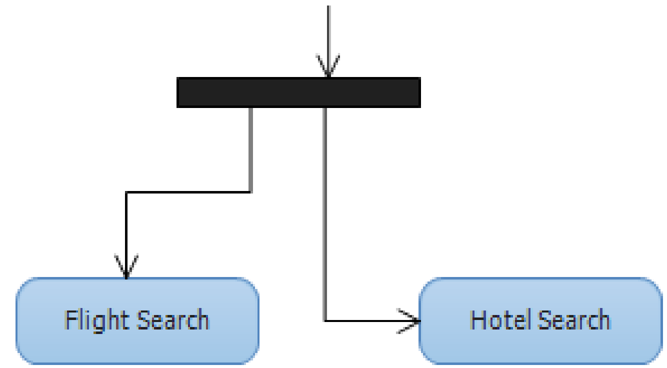
- **Defines concurrency and synchronization**
  - Performance and scalability design options
  - What components can run concurrently?
  - How such concurrent execution is synchronized?

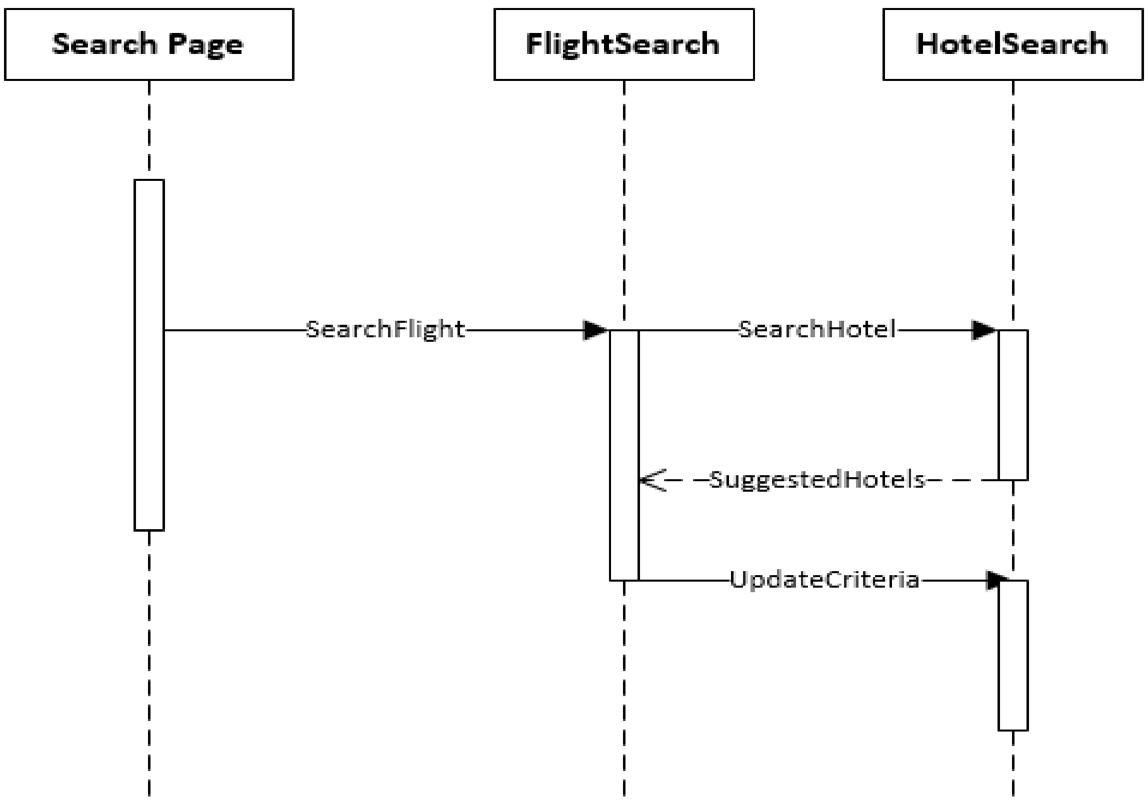
The Process Viewpoint

- **Various models are created:**
  - How system workflows are partitioned across processes?
    - Overall system's process structure
  - Process threads inside each process
  - Map concurrent functions into processes and threads
  - Interprocess communication
    - How concurrent components across separate processes can interact?
    - Ex: Remote Procedure Calls (RPCs) and Named Pipes
  - Runtime states, transitions, and events of concurrent components
  - Synchronization between threads
    - How to prevent concurrent threads from corrupting data?
    - Ex: Shared memory location between threads
    - Ex: Data store data accessed by concurrent threads
    - How to lock and release this data?

Notation


«process»
Flight Search
«process»
Hotels Search

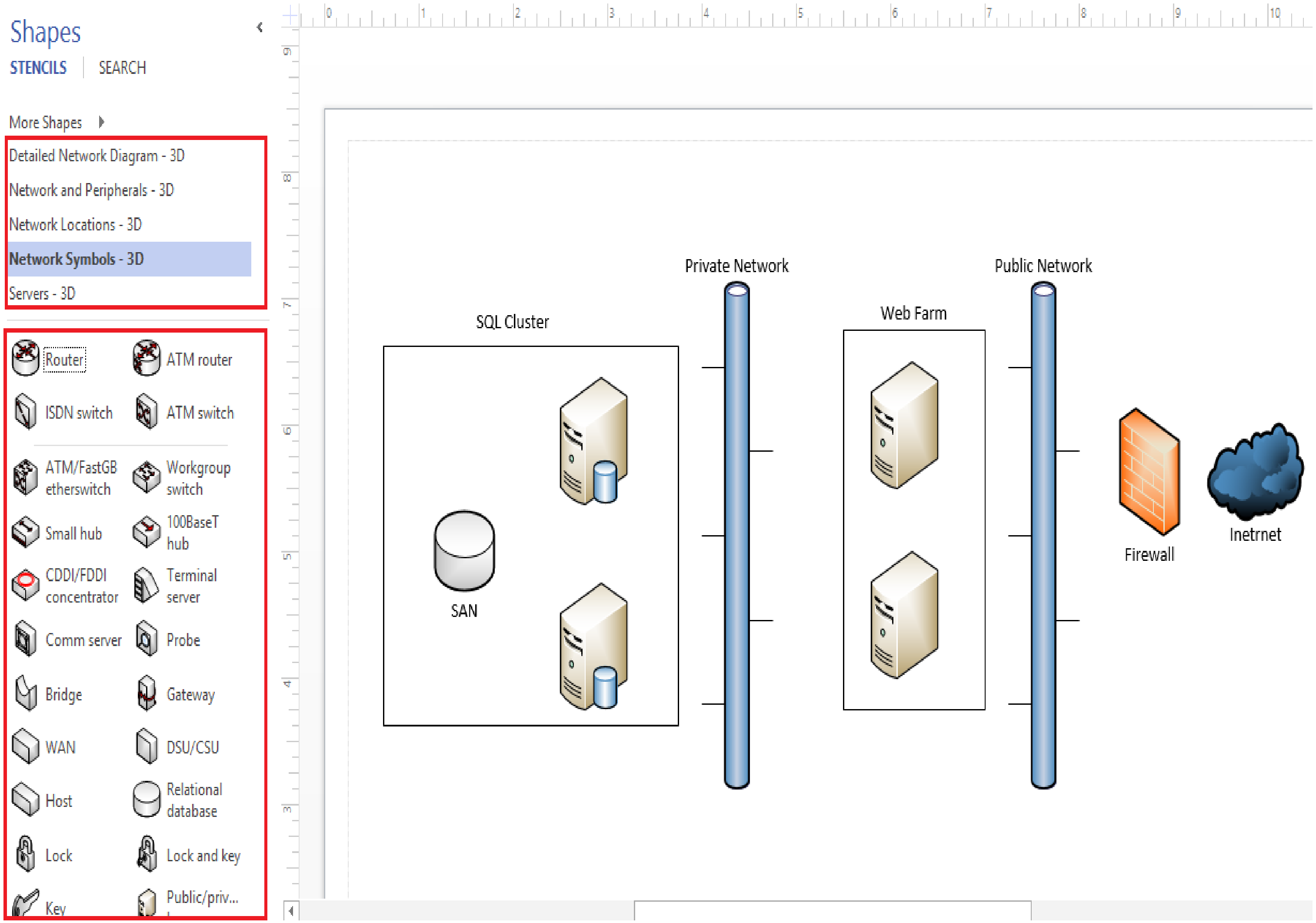

Flight Search
Hotel Search


Search Page
FlightSearch
HotelSearch
SearchFlight
SearchHotel
SuggestedHotels
UpdateCriteria

The Physical Viewpoint

- **The physical environment where the system will run**
  - Servers, firewalls, load balancers, storage facilities, networking, etc…
  - Hardware and network technical specs based on capacity planning and hardware sizing
  - Mapping software components to devices
  - Cabling, power supplies, temperature control, etc…
- **Physical concerns are mostly architectural concerns**
  - Capacity planning and hardware sizing
  - Mapping of software components
- **Other details can be left to detailed design such as model names, power supplies, cabling, etc…**

Notation


Shapes
STENCILS
SEARCH
More Shapes
Detailed Network Diagram - 3D
Network and Peripherals - 3D
Network Locations - 3D
Network Symbols - 3D
Servers - 3D
Router
ATM router
ISDN switch
ATM switch
ATM/FastGB etherswitch
Workgroup switch
Small hub
100BaseT hub
CDDI/FDDI concentrator
Terminal server
Comm server
Probe
Bridge
Gateway
WAN
DSU/CSU
Host
Relational database
Lock
Lock and key
Key
Public/priv...
Private Network
Public Network
SQL Cluster
Web Farm
SAN
Firewall
Inetrnet

Notation

- **Mapping software components to physical nodes:**
  - Tables and matrices
  - UML component diagrams with stereotyping components by physical node names
  - Adding software component names on the physical diagram (shown in the previous slide)

The Development Viewpoint

- **Planning and design of the development environment**
  - Integrated Development Environments (IDEs)
  - Version control and source code structure
  - Task planning tools
  - Task automation
  - Testing tools and test automation
  - Continuous Integration (CI)
  - Build automation

The Operation Viewpoint

- **Operational aspects while system is in production**
  - Maintainability of SLAs in response to faults and change requests
  - Deployment of changes
  - Monitoring for quality attributes
  - Upgrade procedure
  - Configuration Management (CM) strategy
  - Support procedure and support levels
  - Backup and restore procedures
  - Disaster recover procedures

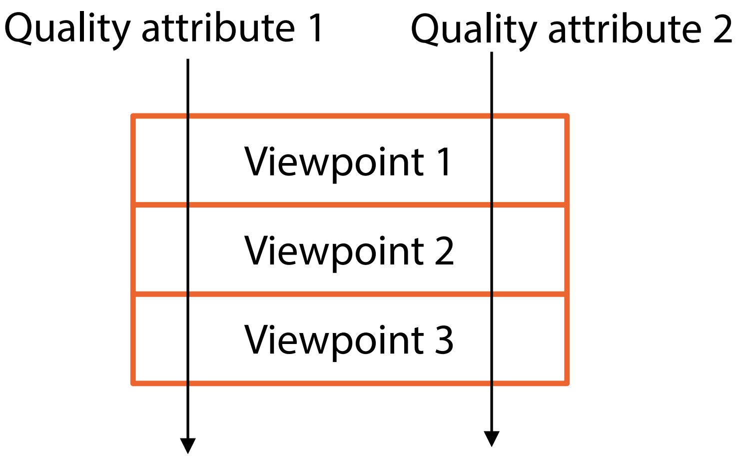
Notation

- **No specific modeling notation**
- **Diagrams, tables, and text**
- **Some UML notations can be useful:**
  - Activity diagrams can document operational procedures (ex: upgrade and backup)
  - Class diagrams can model Configuration Management relationships
  - UML component diagrams can model disaster recover (DR) sites

Other Viewpoints

- **Other viewpoints can be used depending on system type**
- **Hospital management desktop application to be deployed to thousands of client machines**
  - A Rollout viewpoint to handle the rollout concern
- **Big Data applications might require dedicated viewpoints**
- **Business Intelligence applications have high-level business stakeholders**
  - A viewpoint to handle the design of KPIs and dashboards

Apply Quality Attributes



- **Architects work with stakeholders to:**
  - Define the level of which quality attributes are applied
  - Perform tradeoffs between these attributes

Common Quality Attributes

- **Security: privacy, authenticity, access privileges, information protection, etc…**
- **Performance: how the system performs under expected workload**
  - Throughput: maximum number of concurrent requests that can be handled in a unit of time
  - Latency: time required to execute a request – measured in a unit of time
- **Scalability: how performance responds as workload increases**
  - How much resources and cost required to scale-up/scale-out

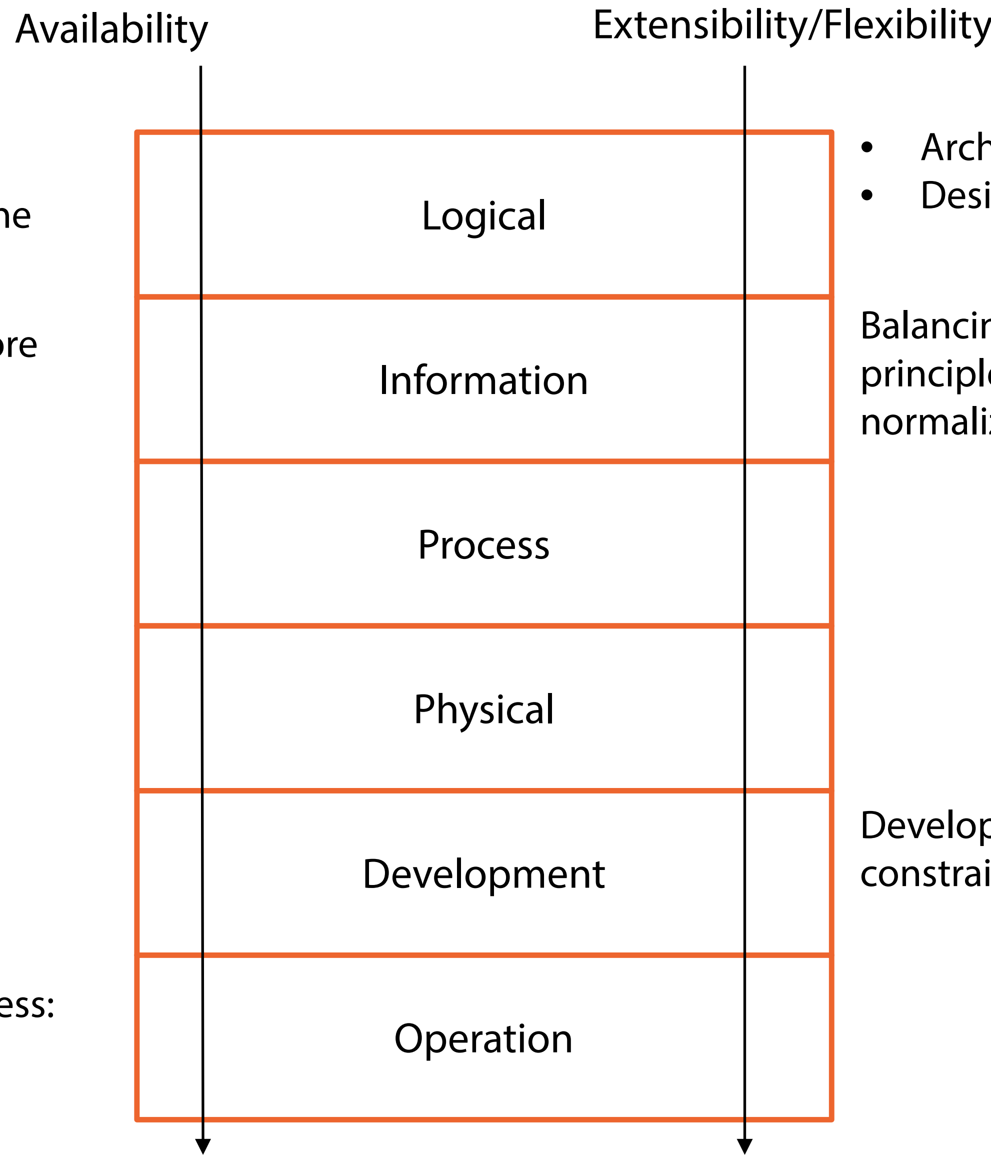
Common Quality Attributes

- **Availability: % time system is capable to perform its intended function**
  - (Total elapsed time – sum of downtime)/total elapsed time
  - Q: Why do we need to balance availability instead of always aiming for max?
  - A: Cost
    - Availability of an internal collaboration server is less important than the availability of the public customer-facing web farm
    - Does it make sense to spend same cost on making both available?
- **Flexibility: How easily can a system be modified and adapted?**
- **Extensibility: Flexibility is achieved by having an Extensible architecture (through design principles)**

Applying Attributes to Viewpoints

Security	Viewpoint	Performance/Scalability
Components that must be secured	Logical	• Relevant components • Parallel programming / asynchronous calls
Information access and transmission security	Information	• Shared data entities • Decentralization
	Process	Concurrent component execution
Hardware and network security	Physical	• Scale-up / Scale-out • Network bandwidth/latency • Transaction Managers
Security development practices and constraints	Development	Development practices and constraints
Security-related production processes	Operation	

Applying Attributes to Viewpoints



Availability:

- Critical components
- Fault-tolerance, offline operation, etc…

Data backup and restore

- Effect of availability
- Ex: Load balancing

- Fault-tolerance
- DR sites
- Backup hardware

Availability-related process:
- Backup/restore
- Failover

Extensibility/Flexibility:

- Architecture Styles
- Design principles (ex: SOLID)

Balancing information design principles (ex: centralization, normalization)

Development practices and constraints

Balance Quality Attributes

- **Achieving the maximum of all attributes is not possible**
  - Tradeoffs must be done
- **Performing tradeoffs:**
  - Dependencies between attributes must be defined
  - Stakeholder engagement drives tradeoffs
  - Stakeholders might change business priorities and attribute levels
- **Recall: This requires close architect/analyst collaboration**

Balance Quality Attributes

- **Keep the following in mind:**
  - Not all quality attributes are related
  - Dependencies must be identified prior to balancing
  - A change in one attribute might be positive or negative to other attributes
  - Tradeoffs is the key technique to balance attributes
  - No single process to apply tradeoffs
  - Tradeoff analysis includes a study of cost vs. benefit

Balance Quality Attributes

	Flexibility	Extensibility	Availability	Performance	Scalability	Security			
Flexibility		+							
Extensibility			+		+	-			
Availability				-					
Performance					+				
Scalability		-		+					
Security		-		-	-			Legend:	
								+	Supports
								-	Hurts
									No Relation

ATAM

- **Architecture Tradeoff Analysis Method**
  - Developed by the Software Engineering Institute (SEI)
- **ATAM "*is a method for evaluating software architectures relative to quality attribute goals*"**
- **ATAM provides "*insight into how those quality goals interact with each other—how they trade off against each other*"**

Validate Architecture

- **Validation is a continuous process and not a one-time activity**
- **We validate the architecture decisions and tradeoffs**
- **Validation might result in:**
  - Change in architecture decisions
  - Change in requirements – as stakeholders understand more about quality attribute tradeoffs

Validate Architecture

- **Validation techniques:**
  - Informal presentations
  - Formal reviews
    - Architecture Description (AD) is reviewed in detail
    - Action plan is set
    - Require: moderator (controller), presenter (author), and reviewers (stakeholders)
  - Prototyping
    - Build a program containing subset of functionality
    - Costly
    - Feasible only if they can justify cost by mitigating risks

Validate Architecture

- **What stakeholders to include in reviews?**
  - Especially a concern for complex systems with various stakeholder groups
- **Depends on tradeoffs and which stakeholder groups are required to reach decisions**

Identify Design Patterns

- **Architectural design is complete!**
- **Now detailed design starts**
- **Design patterns must be identified whenever possible**
  - Ex: Adapter, Decorator, Factory, LazyLoad, and Observer

Detailed Design

- **Inner design of:**
  - Non-architecture-relevant components
  - Last decomposition level of architecture-relevant components
  - Design includes structural and behavioral aspects
- **Detailed data models**
  - Entity-Relationship Diagrams / Class Diagrams
  - How design will be modelled in data stores – such as RDBMS
  - OLAP and Big Data applications require multidimensional and non-structured designs respectively
- **Detailed physical design (model names, cabling details, etc…)**
- **Detailed development considerations (naming conventions, coding standards, documentation standards, coding tools, etc…)**

More Resources

- **UML: Search "UML" in Pluralsight library**
- **Software Architecture:**
  - IASA: http://www.iasaglobal.org/iasa/Certification.asp
  - SEI: http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm
- **ATAM: http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm**
- **Detailed design: Search "design patterns" in Pluralsight library**

Summary

- Identify architecture-relevant scenarios and non-functional requirements
- Scope work and plan iterations and activities
- Define viewpoints that show the solution from different perspectives
- Decompose the architecture into architecture-relevant components
- Identify possible architectural styles

Summary

- **Create views for viewpoints**
  - Logical
  - Information
  - Process
  - Physical
  - Operation
  - Development
  - Others
- **Apply quality attributes**
- **Balance quality attributes**
- **Identify design patterns for detailed design**
- **Perform detailed design**

What's Next?