Software Engineering G22.2440-001

Session 5 – Sub-Topic 3 Presentation Pattern Elicitation Frameworks/Methodologies

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Objectives

- Describe Pattern Elicitation Activities

Roadmap

Definitions

- Solution Architecture
- Component
- Sub-solution/Sub-system
- Architectural Style
- Reference Architecture/Element
- ABASs
- Architectural Mechanisms
- Patterns
- Framework
- Architectural Pattern
- Design Pattern
- Idiom

Becoming a Chess Master

- First learn rules and physical requirements
  - e.g., names of pieces, legal movements, chess board geometry and orientation, etc.
- Then learn principles
  - e.g., relative value of certain pieces, strategic value of center squares, power of a threat, etc.
- However, to become a master of chess, one must study the games of other masters
  - These games contain patterns that must be understood, memorized, and applied repeatedly
- There are hundreds of these patterns

Patterns…

- « Patterns help you build on the collective experience of skilled architects. »
- « They capture existing, well-proven experience in solutions development and help to promote good design practice »
- « Every pattern deals with a specific, recurring problem in the design or implementation of a business solution »
- « Patterns can be used to construct solution architectures with specific properties… »

Becoming a Solution Master

- First learn the rules
  - e.g., the processes/algorithms, data structures and modeling/implementation languages
- Then learn the principles
  - e.g., CBM/SOAD/OOAD methologies, structured/modular/OO/generic programming, etc.
- However, to truly master solution design, one must study the designs of other masters
  - These designs contain patterns must be understood, memorized, and applied repeatedly
- There are hundreds of these patterns

Solution Architecture

- A solution architecture is a description of the sub-solutions and components of a solution and the relationships between them
- Sub-solutions and components are typically specified in different views to show the relevant functional and non-functional properties of a business solution
- The business solution is an artifact
  - It is the result of the solution design activity

Solution Architecture (cont.)

- A set of artifacts (that is: principles, guidelines, policies, models, standards, and processes) and the relationships between these artifacts, that guide the selection, creation, and implementation of solutions aligned with business goals
- Solution architecture encompasses the set of significant decisions about the organization of a solution
  - Selection of the structural elements and their interfaces by which a solution is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger solutions
  - Architectural style or pattern that guides this organization
- Architecture = Elements + Form + Rationale
- Architecture involves a set of strategic design decisions, rules, or patterns that constrain solution design and implementation
  - Architecture constrains design which constrains implementation

Component

- A component is an encapsulated part of a business solution/software system
  - A component has an interface
- Components serve as the building blocks for the structure of a business solution/system.
- At a modeling language level, components may be represented as processes, services, etc
- At a programming-language level, components may be represented as modules, classes, objects or as a set of related functions

Sub-solutions/Sub-systems

- A sub-solution/sub-system is a set of collaborating components performing a given task
- A sub-solution/sub-system is considered a separate entity within a solution/software architecture
  - It performs its designated task by interacting with other sub-solutions/sub-systems and components…

Architectural Style

- An architectural style is a description of component types and their topology
- It also includes a description of the pattern of data and control interaction among the components and an informal description of the benefits and drawbacks of using that style
  - Architectural styles are important engineering artifacts because they define classes of designs along with their associated known properties
  - They offer experience-based evidence of how each class has been used historically, along with qualitative reasoning to explain why each class has its specific properties

ABASs

- Attribute Based Architectural Styles (ABASs)
- ABASs build on architectural styles to provide a foundation for more precise reasoning about architectural design by explicitly associating a reasoning framework (whether qualitative or quantitative) with an architectural style
- These reasoning frameworks are based on quality attribute-specific models, which exist in the various quality attribute communities (such as the performance and reliability communities)

Back to “Patterns”

- A common solution to a common problem in a context
- Examples:
  - Architectural Patterns
  - Design Patterns
  - Frameworks
  - Implementation Patterns
  - Idioms

Reference Architecture/Element

- A reference architecture is a self-contained architectural style
  - i.e., provides a description of readily applicable set of component types and their topology
  - e.g., SOA, OMA, etc.
- A reference element is an atomic constituent of a reference architecture
  - e.g., various categories of SOA services, OMA components, etc.

Framework

- A set of assumptions, concepts, values, and practices that constitutes a way of viewing the current environment
- Defines the general approach to solving the problem
- Skeletal solution, whose details may be analysis/design patterns
- A solution/software framework is a partially complete solution/software (sub-) system that is intended to be instantiated
  - It defines the architecture for a family of (sub-) systems and provides the basic building blocks to create them
  - It also defines the places where adaptations for specific functionality should be made

Architectural Mechanism

- An architectural mechanism represents a common solution to a problem occurring a number of places in the system
  - Often a frequently-encountered problem → a pattern
- Provide a clean way to “plug” technology-based solutions into technology-independent application models
  - Examples
    - Commercial off-the-shelf products
    - Database management systems
    - Distributed access (networking, remote methods, etc.)
    - Enterprise platforms (Microsoft .NET, Java 2 Enterprise Edition, CORBA Services, etc.)
- Three categories
  - Analysis mechanisms (conceptual: persistence, remote access, etc.)
  - Implementation mechanisms (concrete: e.g. RDBMS, J2EE)
  - Product selection mechanisms (actual: Oracle, BEA WebLogic)

Architectural Pattern

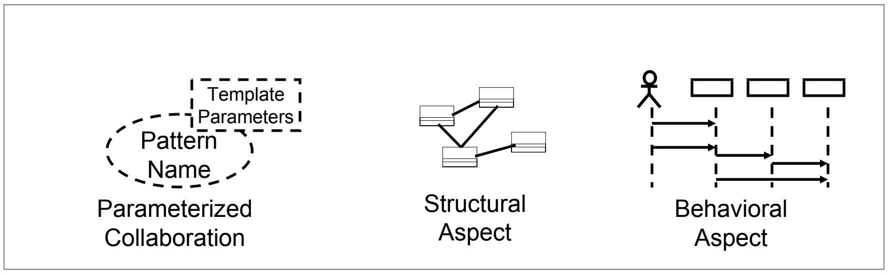
- An architectural Pattern expresses a fundamental structural organization schema for software systems
  - Predefined subsystems and their responsibilities
  - Rules and guidelines for organizing the subsystems
  - Relationships between subsystems
- Examples of architecture patterns
  - Layers
  - Model-View-Controller (separate the user interface from underlying application model, integrated by a controller)
  - Pipes and filters (Unix, signal processing systems, etc.)
  - Shared data (web-based and corporate information systems with user views of a shared RDBMS)

Design Pattern

- A solution to a narrowly-scoped business/technical problem
  - A fragment of a solution, a partial solution, or a piece of the puzzle
  - A solution to a common design problem
    - Describes a common design problem
    - Describes a proven solution to the problem
    - A solution based on experience
  - Design patterns discuss the result and trade-offs of applying the pattern
  - Design patterns provide the capability to reuse successful designs
- A design pattern provides a scheme for refining the subsystems or components of a solution, or the relationships between them

Design Pattern (cont.)

- A technical pattern is modeled as a parameterized collaboration in UML, including its structural and behavioral aspects



Idiom

- An Idiom is a low-level pattern specific to an implementation (i.e., execution/programming) language
- An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language

Roadmap

EAMF Framework

- Introducing the EAMF Framework
- EAMF Framework Concepts
- EAMF Standard Structural Elements

Spelling it Out!

- **E**nterprise
- **A**rchitecture
- **M**anagement
- **F**ramework

Why EAMF?

- Design Information is typically not organized well enough
  - Cannot find documentation when you need it
- Solutions to difficult problems are typically too complex
  - Documentation, when available, is difficult to understand and buried in long and arcane documents
- Communication of problems and their solutions is typically inefficient
  - Long and boring meetings with too many people.

EAMF Objectives

- Identify, categorize, capture and catalogue optimal solutions and their intent based on best practice architectural analysis and design guidelines, and patterns
- Divide and conquer problem and solution spaces
- Formalize architectural processes
  - From the inception of a problem
  - To the deployment of a solution
  - And every step in-between
- Encourage top-down, bottom-up, and abstract thinking
- Simplify interactions between architects

EAMF Practical Ingredients

- EAMF Framework
  - Supporting structure used as a container to present and maintain EAMF artifacts
  - EAMF artifacts include reference and/or solution specific pattern clusters along with their intent
- EAMF Methodology
  - Set of documented architectural analysis and design procedures and guidelines used to produce EAMF artifacts stored in the EAMF Framework

Introducing the EAMF Framework

- The EAMF Framework subsumes concepts and elements to store captured information
  - Framework Concepts
    - Organization techniques used to capture architecture information
  - Structural Elements
    - The framework includes general purpose structural elements that may be combined for a specific purpose

EAMF Framework Concepts

- Map
- Landscape
- Grid
- Perspective
- ViewPoint
- View
- Cell
- Artifact
- Level of Abstraction
- Level of Encapsulation

EAMF Framework Concepts: Map

- The EAMF map represents the underlying (virtual) area on which all the EAMF artifacts are located independently of the perspectives and views used to create and/or visualize them
- The EAMF concept of a map accommodates back and forth navigation across disciplines and viewpoints within a given perspective
- The EAMF concept of a map facilitates re-engineering or pre-existing architectures
  - EAMF is designed to support architecture engineering, re-engineering, and evolution modes

EAMF Framework Concepts: Landscape

- The EAMF architectural landscape encompasses all the architectural elements that exist at a given time on the EAMF map
- Effectively, it should be possible to look at an existing architectural landscape and map its artifacts to various EAMF perspectives
  - This type of effort is referred to as architectural re-engineering

EAMF Framework Concepts: Grid

- The EAMF grid is a two-dimensional table
- Its columns represent perspectives and its rows represent views
- The grid is the main structure used in EAMF architecture engineering mode to capture information about a problem and characterize its optimal solution
- Each grid is specific to a particular architecture domain

EAMF Framework Concepts: Perspective

- A perspective is a window into the architectural sub-domains that make up a complete architecture
- Example: Enterprise Architectures
  - Architectural sub-domains include business, information, application, and technology architectures
  - EAMF perspectives may be defined accordingly to focus on one or more architectural sub-domain

EAMF Framework Concepts: Viewpoint

- A viewpoint is a window within a perspective that focuses on a very specific aspect of an architecture sub-domain
  - A viewpoint and a perspective are conceptually related as a viewpoint is more specific than a perspective
  - Viewpoints encompass important information about a problem context within a given perspective that is not comprehensive enough to describe the whole problem context
- A viewpoint is a decomposition tool that helps architects focus on one important aspect of the problem rather than looking at the problem from many angles at the same time

EAMF Framework Concepts: View

- Viewpoints in a perspective divide that perspective in further detail areas
- A view, in contrast, looks at the same perspective with different intentions in mind
  - A view is intention related while a viewpoint is area related

EAMF Framework Concepts: Cell

- A cell may contain any number of artifacts of different types

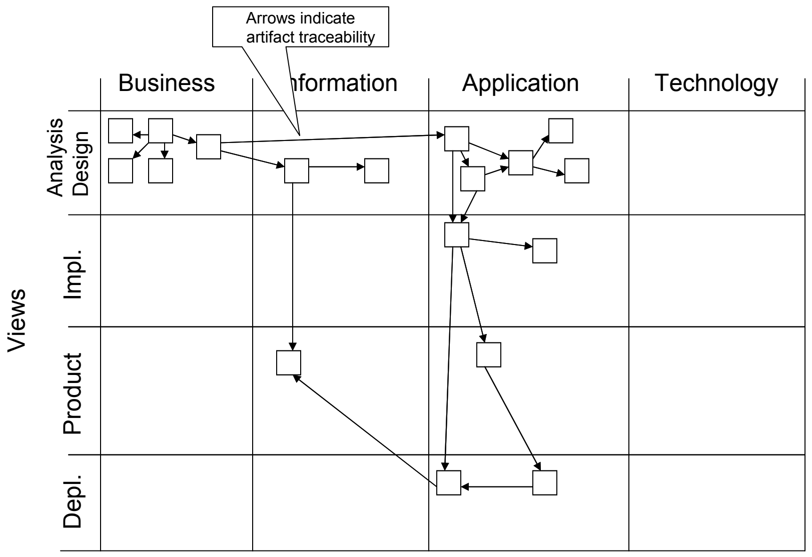
EAMF Framework Concepts: Artifact

- Each cell in the perspective-view grid is populated with artifacts that together capture the following information for a given business problem:
  - Complete static and dynamic views
  - Solution structure and information as to how that solution structure is achieved
  - Catalogs that help designers create new solutions structures
- As-is (current state)
- To-be (future state)

EAMF Framework Concepts: Artifact (cont.)

- Three types of artifacts
  - Textual
    - Documents, notes, e-mails, memos, anything that captures information in plain text.
  - Graphical
    - Images, graphs, diagrams, pictures, etc.
  - Tabular
    - Tables, matrices, spreadsheets, etc.

EAMF Framework Concepts: Artifact (cont.)



EAMF Framework Concepts: Level of Abstraction

- EAMF levels of abstraction on graphical artifacts
  - The graphical artifacts have conceptual, logical and physical levels of abstraction, where conceptual is very high level and does not contain any specific detail information
    - The physical level is a very specific and detailed level, and the logical level somewhere between the conceptual and physical level
- EAMF levels of abstraction on tabular artifacts
  - Levels of abstractions are organized on the vertical axis of the matrices (rows) where the top rows represent high level, more abstract views and the bottom rows represent lower level, highly detailed views
  - EAMF defines different levels of abstractions for each perspective and each view within the perspective

EAMF Framework Concepts: Level of Encapsulation

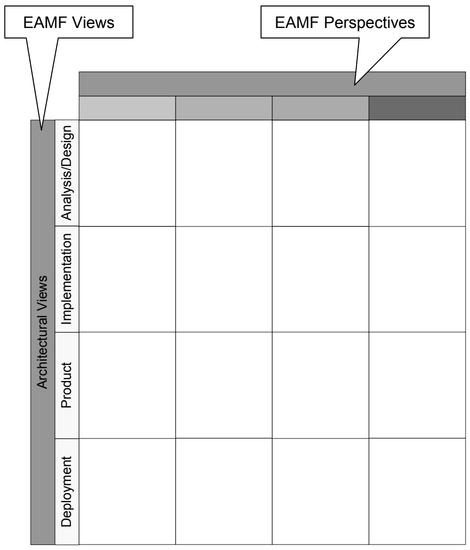
- Levels of encapsulation/nesting may apply to certain artifacts
  - For example, a pattern hierarchy may be used to describe the organization of a collection of patterns that make up a framework in a given view
    - In this case the patterns involved in the hierarchy are at the same level of abstraction (i.e., they are an exclusive part of either of the analysis, design, or implementation view)
    - Pattern languages may suggest how multiple pattern can be used together to make up a framework, without always implying levels of encapsulation

Roadmap

EAMF Standard Structural Elements

- EAMF Perspectives
- EAMF Views
- Perspective-View Grid
- Catalog
- Standard Matrices
- Domain Specific Matrices
- Standard vs. Domain Specific Matrices
- Constraints and Restrictions Document
- Capability Matrix

EAMF Perspectives



- The EAMF framework relies on an architectural decomposition in terms of four perspectives
  - Business
  - Application
  - Information
  - Technology
- EAMF uses a 4x4 grid where perspectives are placed horizontally and views are placed vertically

EAMF Perspectives (cont.)

- Business Perspective
  - This perspective establishes a context around the problem from a business point of view
- Information Perspective
  - This perspective establishes a context around the problem from a data structure solution point of view
- Application Perspective
  - This perspective establishes a context around the problem from the software solution point of view
- Technology Perspective
  - This perspective establishes a context around the problem from the hardware solution point of view

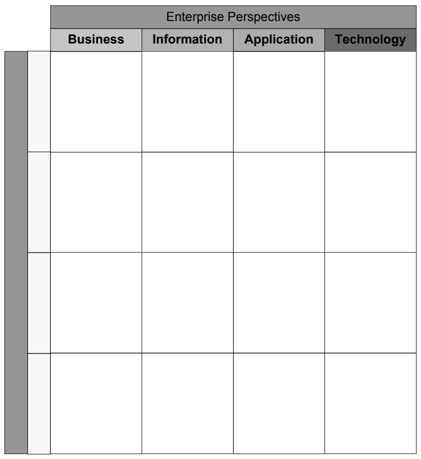
Business

EAMF Views

- Analysis and Design View (Model View)
  - This view defines reference architectures, architectural styles and patterns. The problem analysis and design model artifacts at conceptual and logical levels are placed in this view
- Implementation View
  - This view defines reference implementations, implementation styles and patterns
- Product View
  - This view holds artifacts that list the physical products used in the solution. The products are different for each perspective.
- Deployment View
  - This view holds artifacts that show how the solutions are used in a physical environment with the determined products.

Perspective-View Grid

- The EAMF perspectives and views create a 4x4 grid where perspectives are placed horizontally and views are placed vertically as shown
- This grid is the primary mechanism through which all architectural analysis, design, implementation and deployment information is stored on the map in an organized manner



Catalog

- A catalog exists in EAMF independent of a problem context
  - There can be one catalog per view per perspective
- A catalog contains best practice patterns available in an industry
  - For practical reasons, catalogs only include pattern information relevant to a specific domain
  - The catalogs are maintained by BAs and TAs to keep track of emerging and evolving patterns

Standard Matrices

- The *standard matrices* show how a particular problem should be solved in an ideal environment
- Most of the time a company does not have the ideal environment to implement the solution
  - These matrices can be used as a description of the desired state of architecture for a particular problem
  - They, too, need to be updated continuously to reflect recent changes in the patterns world

Domain Specific Matrices

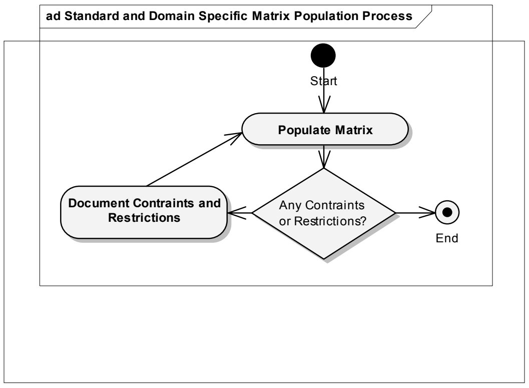
- The *domain specific matrices* capture the solution implemented in an environment that is less than ideal
  - There might be many reasons why a company does not have an ideal environment which is mostly financial and time constraints related
- Domain specific matrices are a snapshot that shows the current architecture and implementation
  - When the company environment changes towards the ideal environment the solution can be re-factored and brought closer to the solution that is described in the standard matrices

Standard vs. Domain Specific Matrices

- The standard matrix shows an ideal choice of patterns for the solution while the domain specific matrix shows choice of patterns that need to be used (i.e., mechanisms) in a company domain based on the restrictions and constraints imposed either by the company or by external forces
- Standard and Domain Specific Matrices can respectively be considered as the future and current state matrices
  - Standard Matrix represents the future state because it captures information for an ideal solution
    - Most often, due to constraints and restrictions, internal or external, this ideal solution cannot be realized
  - Domain Specific Matrix represents the current state solution
    - Therefore, the information in these matrices can be used as gap analysis and provide executive management with important information for decision making purpose
- Capturing the constraints and restrictions during the process also ties historical decisions to the current state.

Standard vs. Domain Specific Matrices: Population Process

- The following diagram depicts the processing rule which applies every time when there is a standard and domain specific matrix independently from the enterprise perspective:


ad Standard and Domain Specific Matrix Population Process
Start
Populate Matrix
Document Contraints and Restrictions
Any Contraints or Restrictions?
End

Constraints and Restrictions Document

- As explained in the previous section if there is a need to create a domain specific matrix, the reasons must be captured within a Constraints and Restrictions Document
- The document specifically outlines why a standard matrix or a previously used domain specific matrix cannot be used
- Instances of this document are repeated wherever there is a need to replace a standard matrix with a domain specific matrix

Capability Matrix

- A capability matrix shows the capabilities of an entity as a tabular artifact
- This matrix type is mostly used in the application perspective for determining the functional capabilities of reference architectures, reference implementations and products
  - However, there are other perspectives where the capability matrices are employed
- The format of the matrix depends on the type of capabilities it captures.

Roadmap

EAMF for Business Architects

- Using EAMF Standard Structural Elements in the Business Perspective

Business Perspective: Business Model

- Business Process
  - A long running set of actions or activities performed with specific business goals in mind
    - Business processes typically encompass multiple service invocations
    - Examples of business processes are: *Initiate New Employee*, *Sell Products or Services*, and *Fulfill Order*
  - In SOA terms, a business process consists of a series of operations which are executed in an ordered sequence according to a set of business rules
    - The sequencing, selection, and execution of operations is termed service or process *choreography*
      - Typically, choreographed services are invoked in order to respond to business events.

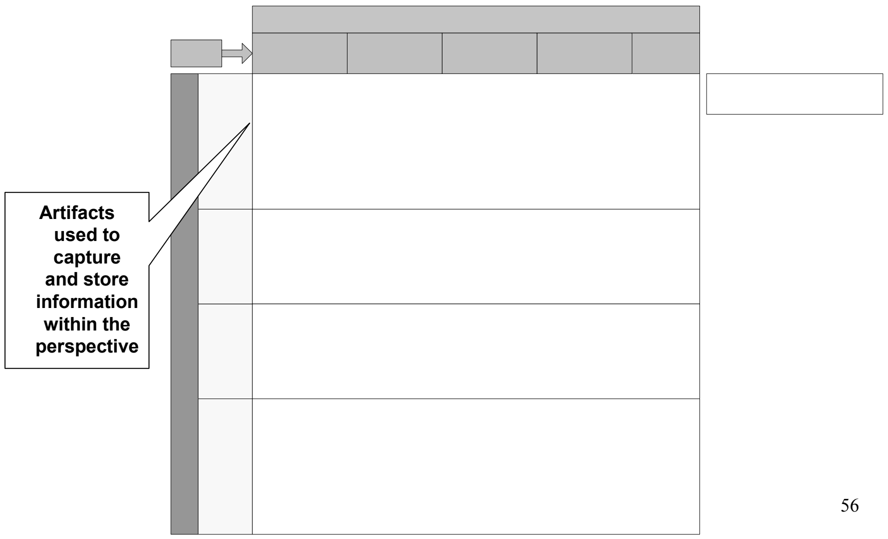
Business Perspective: Business Patterns

- Choreography
  - A choreography is the observed sequence of messages exchanged by peer services when performing a unit of work
  - Services do not need to be orchestrated to perform a unit of work (this is a concept that emerged and should have stayed in the last century)
    - This is a very common misconception, actually most units of work are accomplished by a series of "orchestrated services" performing a choreography
    - There are several industry efforts in the area of choreography languages, such as BPML (defined by BPMI.org), BPSS (defined by ebXML), IBM's WSFL, Microsoft's XLANG, and IBM/Microsoft/BEA's BPEL4WS and their companion specifications WS-Coordination and WS-Transaction, etc.

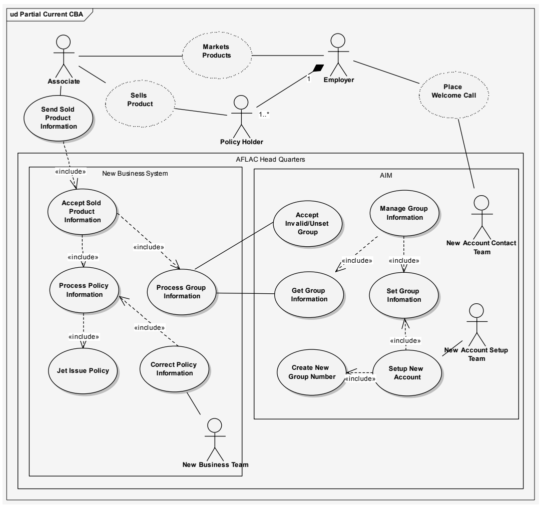
Business Perspective: Business Patterns

- Orchestration
  - An orchestration is a generalization of composition that sequence services and provide additional logic to process data that does not include data presentation
  - The same language can be used to perform a complex unit of work achieved by invoking a series of service operations
  - Any given orchestration is not forced to expose a service interface
  - If it does, it is a composition
- An orchestration is executed by an orchestration engine
  - BPEL is an orchestration programming language

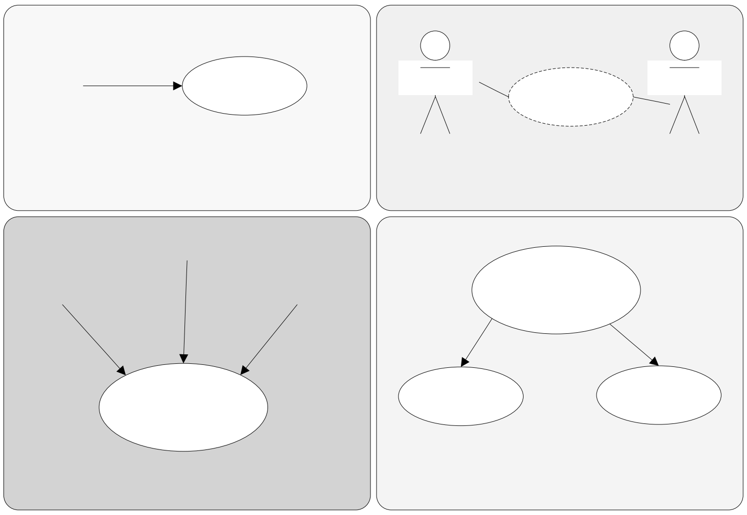
Business Perspective: Filling up the Business Grid


Artifacts used to capture and store information within the perspective

Business Perspective:
Conceptual Business Architecture (CBA)



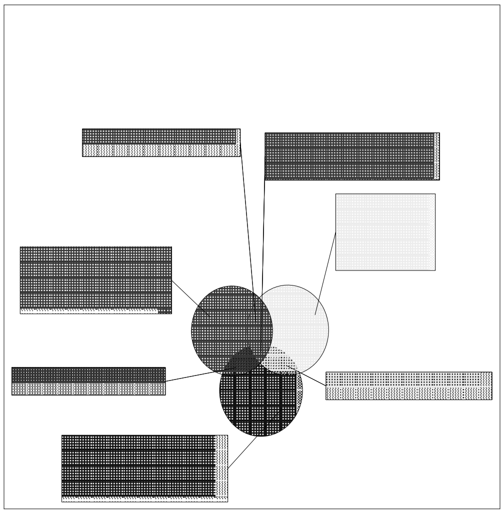
Business Perspective:
Problem Specific CBA



Business Perspective: Viewpoints

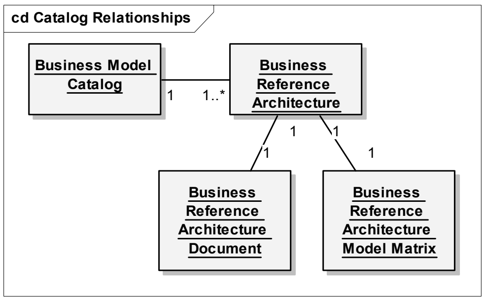
- Process
  - This viewpoint isolates the set of involved business processes
- Process Interoperability
  - This viewpoint isolates the unique relationships between involved business processes
- Process Organization Interface
  - This viewpoint isolates the relationship between the business processes and their interaction with the people
- Organization
  - This viewpoint isolates the people around the business problem and its solution structure
- Location
  - This viewpoint isolates the location of the people in the context of the business problem and its solution structure

Business Perspective: CBA-Viewpoints Relationship



Business Perspective: Business Model Catalog

- Business Model Catalog is a problem independent artifact in EAMF
- It lists the available Business Reference architectures and more detailed information for each of these business reference architectures


cd Catalog Relationships
Business Model Catalog
1
1..*
Business Reference Architecture
1
1
1
1
Business Reference Architecture Document
Business Reference Architecture Model Matrix

Business Perspective: Reference Architectures Model Matrix

- Combined with the Business Perspective viewpoints the Model Matrix looks like the following tabular artifact
- There will be one populated instance of the above matrix per business reference architecture

Business Perspective:

Standard and Domain Specific Model Matrices

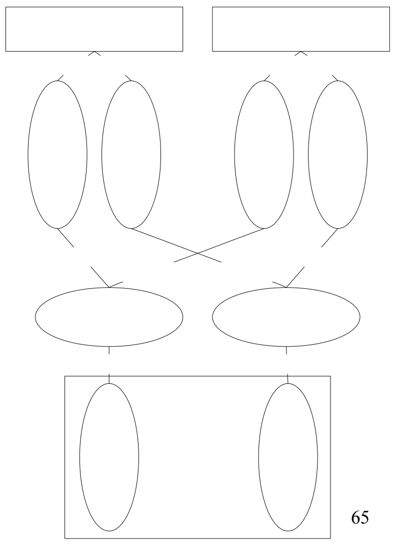
- Standard and domain specific model matrices look exactly like the catalog model matrix, except there will be one or more instances per problem description
- While a catalog shows all applicable styles and patterns for that reference architecture a standard model matrix instance shows only the patterns that apply to the ideal solution of that problem
- Domain Specific Model Matrix instance shows only the patterns that apply to a solution imposed by the constraints and restrictions

Business Perspective: Capabilities and Requirements Matrix

- Capabilities and Requirements Matrix (a.k.a., CR Matrix) is used to capture the non-functional and functional, business and technical capabilities to solve a specific business problem
- Capabilities are set of concerns that a solution tries to address
- Requirements are entries in the matrix that describe how a solution meets a specific business requirement while addressing the capability

Business Perspective: Adding a New Reference Architecture

- EAMF is not complete in terms of standard capability sets
- EAMF must be augmented when the experience level of the architects reach an expert level in a reference architecture such as SOA, ECM, BPM and so on
- The process of determining a more abstract capability sets is shown on the right



65

Business Perspective: Non-Functional Capabilities (NFCs)

- Non-functional capabilities are filled by BAs and TAs in collaboration. There are three types of non-functional requirements
- Only the project-based capabilities are answered when filling out the CR-Matrix
- Organizational and external non-functional capabilities represent constraints and are used when determining domain specific solutions

- Non Functional Capabilities (NFCs)
  - Project-Based NFCs
    - Accuracy
    - Availability
    - Efficiency
    - Extensibility - Upgradeability – Modifiability – Adaptability- Flexibility
    - Interoperability
    - Portability
    - Recoverability
    - Reliability – Dependability
    - Reusability
    - Scalability – Capacity
    - Security – Accessibility – Anonymity- Vulnerability
    - Usability – Operability
  - Organizational NFCs
    - Readability – Simplicity – Understandability
    - Maintainability
    - Testability – Verifiability
    - Traceability
  - External NFCs
    - Ethical
    - Legislative (Privacy – Safety)
    - Planning (Cost, development time)

66

Business Perspective:
Project-Based NFCs

- Accuracy
  - Accuracy is the quantitative measure of the magnitude of error
- Availability
  - Degree to which a service, system or component is operational and accessible when required for use
- Efficiency
  - Efficient is the degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks)
- Extensibility - Upgradeability – Modifiability – Adaptability-Flexibility
  - Adaptability
    - Ease with which software satisfies differing system constraints and user needs.
  - Flexibility
    - Eease with which a system can be modified for use in applications other than those for which it was specifically designed.
    - Use Strategy Pattern, etc…
      - Interoperability
      - Portability

Business Perspective:
Project-Based NFCs (cont.)

- Interoperability
- Portability
  - Ease with which a system or component can be transferred from one hardware or software environment to another
- Recoverability [IEEE 90]
  - Recoverability is the restoration of a system, program, database or other system resource to a prior state following a failure or externally caused disaster
- Reliability – Dependability
  - Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time
- Reusability

Business Perspective:
Project-Based NFCs (cont.)

- Scalability - Capacity
  - Scalability is the ease with which a system or component can be modified to fit the growing problem area. Scalability has two variants, hardware and software
- Security – Accessibility – Anonymity- Vulnerability
  - Security is the ability of a system to manage, protect and distribute sensitive information
- Usability - Operability
  - Usability is the ease with which a user can learn to operate, prepare inputs for and interpret outputs of a system or component

Business Perspective:
Organizational NFCs

- Readability – Simplicity – Understandability
  - The degree to which a system's functions and those of its component statements can be easily discerned by reading the associated source code
- Maintainability
  - The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment
- Testability – Verifiability
  - The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
- Traceability
  - The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another
- Delivery
- Implementation

Business Perspective:
External NFCs

- Ethical
  - The policies regarding ethics that a corporation adopts when conducting business
- Legislative (Privacy – Safety)
  - The policies that a corporation is imposed upon when conducting business
- Planning (Cost, development time)
  - The resource constraints that are imposed on a project

Business Perspective:
Sample Functional Capabilities

- EAMF divides the functional capabilities in three distinct top level categories:
  - OMA Specific Services
    - Concurrency Service (http://www.omg.org/docs/formal/00-06-14.pdf)
    - Externalization Service (http://www.omg.org/docs/formal/00-06-16.pdf)
    - Event Service
    - Interface Invocation Service
    - Life Cycle Service
    - Naming and Directory Services (http://www.omg.org/docs/formal/04-10-03.pdf)
    - Notification Service
    - Persistence State Service
    - Security Service
    - Trading Object Service
    - Transaction Service
  - OMA Specific Facilities
  - OMA Application Objects

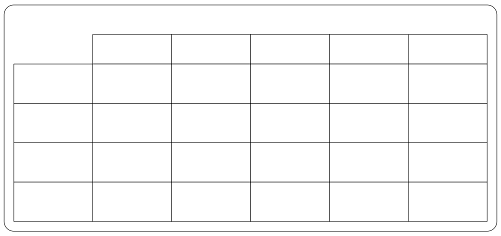
Roadmap

EAMF Methodology

- Using a PDA EAF
- NFR-Driven Pattern Elicitation

Using a PDA EAF

- Gather problem definition – Business Requirements
- Create Conceptual Business Architecture Diagrams
- Create Business Catalogs
  - Business model matrix (BMM) captures reusable business reference architectures, architectural styles and patterns
  - Business implementation matrix (BIM) captures reusable reference implementations, styles and implementation patterns
  - The implementation view is prescriptive and the model view is descriptive
- Run Through Decomposition Process
  - Populate Standard and Domain Specific Business Model Matrices



Using a PDA EAF (continued)

- Populate Capabilities and Requirements Matrix (CR-Matrix)
  - Before the architects start creating instances of CR-Matrices several questions must be answered:
    - What is the primary viewpoint for this business problem?
    - Which patterns are related to each other across viewpoints?
    - Which patterns or styles do not contribute to a technology solution pattern?
    - Let us assume that the answers to the above questions are:
      - The primary viewpoint is **Process.**
      - The related patterns across viewpoints are **EBP.Producer.LowVolume** and **C2B.RequestResponse.FastAccess**.
      - The **GroupsOfIndividuals** and **Centralized** styles do not contribute to the business problem solution .
    - Then
      - Create one CR-Matrix instance for each pattern in the primary viewpoint.
      - Create one CR-Matrix instance for each set of related patterns across viewpoints, and do not include non-contributing patterns.
      - As follows:
        - EBP.Producer.LowVolume – C2B.RequestResponse.FastAccess
        - EBP.Transformer.HighVolume.
    - Policies are entered into the CR-Matrix

Using a PDA EAF (continued)

- Using a CR-Matrix
  - Generation of the Conceptual Technology Architecture Diagram
  - Identification and Confirmation of Appropriate Reference Architecture(s)
    - Generation of a Logical Architecture Analysis Diagram
    - Generation of the Analysis Model
    - Identification of Applicable Pattern(s)
    - Generation of a Logical Architecture Design Diagram
    - Generation of the Design Model
  - Identification of Applicable Reference Implementation(s)
    - Identification of Applicable Implementation Pattern(s)
    - Refinement of the Logical Architecture Design Diagram
    - Refinement of the Design Model
- Product Mapping
- Deployment
- Working with Developer
- Deployment Mapping

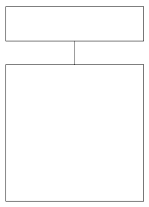
NFR-Driven Pattern Elicitation

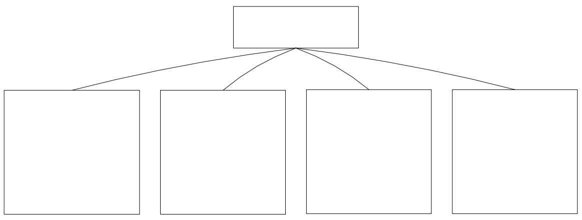
- NF policies identified in the CR-Matrices are used to determine appropriate styles and patterns for the solution
  - Not all non-functional capabilities lead to software design patterns
  - Some non-functional capabilities such as reliability, availability, recoverability and dependability (and possibly others) require hardware deployment patterns along with organizational behavior changes towards quality
- There is no (bullet-proof) defined process that would help an architect to identify appropriate patterns
  - A pattern can be applicable to a certain problem but it may not be appropriate
  - The identification of the applicable and appropriate pattern requires immense working pattern knowledge which is not only knowing and understanding what patterns are but also recognizing when and when not to use certain patterns
  - NFR Framework Based Approach is suggested

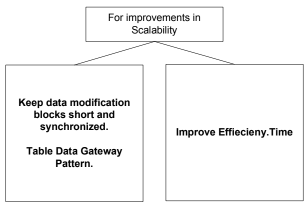
NFR-Driven Pattern Elicitation: Sample Guidelines

EBP.Producer.LowVolumeLowFrequency - C2B.RequestResponse

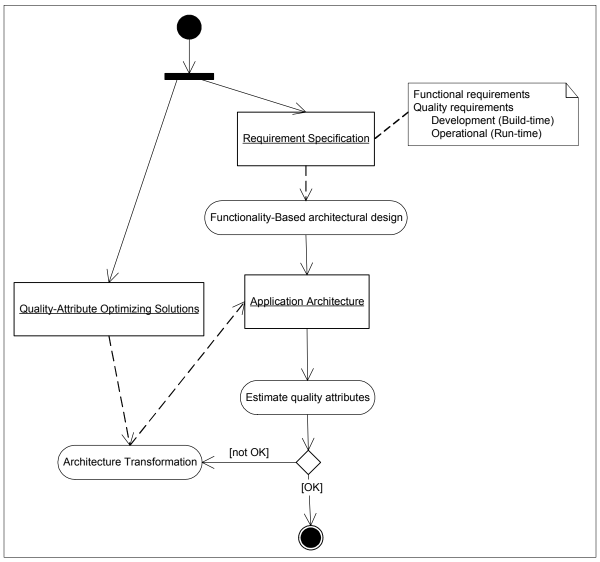
Policy Type / Motivation	Measurable (only verifiable after implementation)	Concrete (based on proven design)
Business Driven - Required - (resulting from analysis)	**Efficieny.Time** : 8 sec/per request/per user **Scalability** : 20 concurrent users to 40 concurrent users without software modifications	**Availability** : 6:00 am - 6:00 pm Business days **Security** : Only authenticated and authorized AFLAC employees Concurrency Persistent State Security Transaction
Technology Driven - Desired - (resulting from design considerations)	**Extensibility** : Same generation process with different algorithms will be supported. **Efficieny.Time** : 0.5 sec/per request/per user **Readability** : Follow company standards	**Testability** : Provide test harness, debugging and adjustable levels of logging capabilities. Naming and Directory






For improvements in Scalability
Keep data modification blocks short and synchronized.
Table Data Gateway Pattern.
Improve Effiecieny.Time

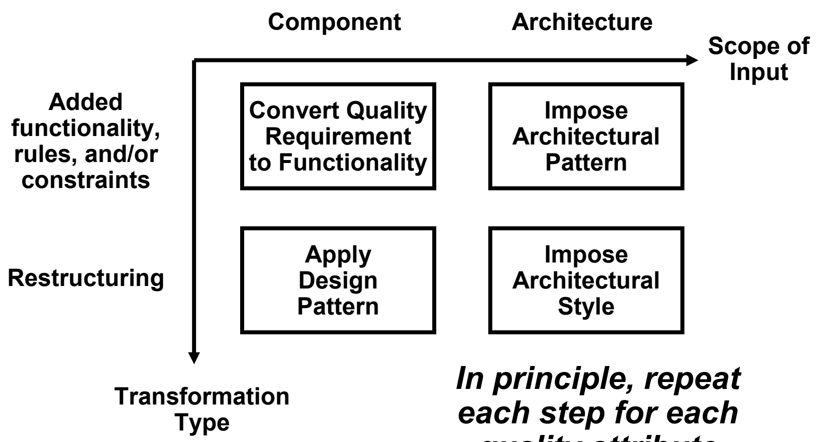
Architecture Design Method


Functional requirements
Quality requirements
Development (Build-time)
Operational (Run-time)
Requirement Specification
Functionality-Based architectural design
Quality-Attribute Optimizing Solutions
Application Architecture
Estimate quality attributes
Architecture Transformation
[not OK]
[OK]

Overview of Architecture Design Method

- Functionality-based architectural design
  - Define system context
  - Identify archetypes (core functional abstractions)
  - Decompose into components
    - Interfaces, domains, abstraction layers, domain entities, archetype instantiations
  - Describe system instantiations
- Assess quality attributes
  - Define quality profiles
  - Scenario-based assessment
  - Simulation
  - Mathematical modeling
  - Experience-based assessment
- Architecture transformation
  - Impose architectural style
  - Impose architectural pattern
  - Impose design patterns
  - Convert quality requirements to functionality
  - Distribute requirements

Architecture Transformation Categories



Component
Architecture
Scope of Input
Added functionality, rules, and/or constraints
Restructuring
Convert Quality Requirement to Functionality
Impose Architectural Pattern
Apply Design Pattern
Impose Architectural Style
Transformation Type

In principle, repeat each step for each quality attribute

Convert Quality Requirements to Functionality

- Self-monitoring
- Redundancy
- Security
- etc.

Distribute Requirements

- Distribute system-level quality requirements to the subsystems and components
  - System quality X; component quality x_i
  - $X = x_1 + x_2 + x_3 + \ldots + x_n$
- Separate functionally-related qualities
  - e.g., Fault-tolerant computation + fault-tolerant communication

Impose an Architectural Style

- Bass *et al.*, Attribute-Based Architectural Styles (ABASs)
  - Pipes and filters
  - Layers
  - Blackboard
  - Object-Orientation
  - Implicit Invocation
  - etc.

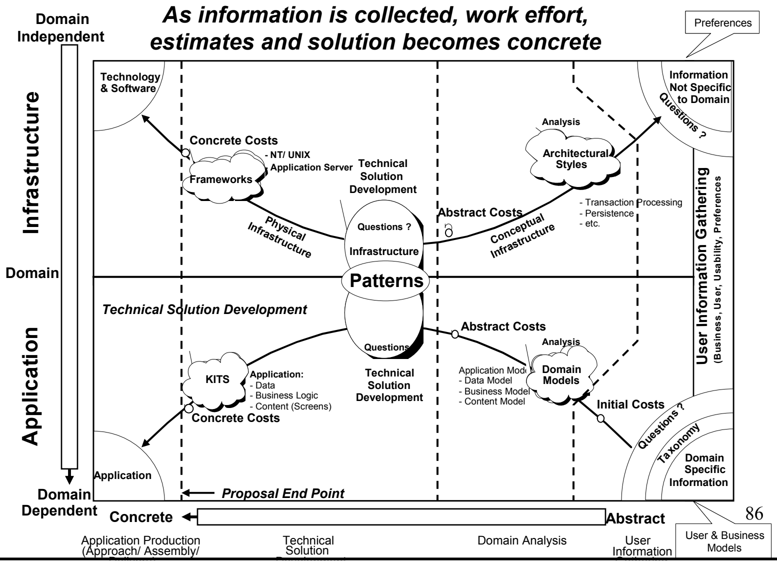
Impose an Architectural Pattern

- Concurrency
  - Processors, processes, threads, scheduling, etc.
- Persistence
  - Database management system, application-level persistence and transaction handling, etc.
- Distribution
  - Brokers, remote method invocation, etc.
- Graphical User Interface
  - MVC, PAC, etc.

Apply a Design Pattern

- Gang-of-Four, Buschmann, etc.
  - Façade
  - Observer
  - Abstract Factory
  - etc.

Meta-Tools For Model Driven Architectures



Pattern Matching Approach

http://www.scs.carleton.ca/~weiss/research/nfr/cito.pdf

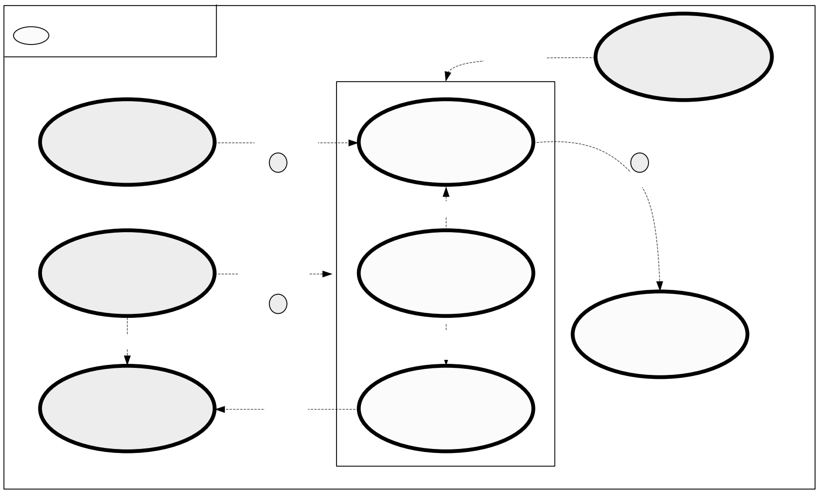
- Represent force hierarchies that match a problem domain using GRL notation
- Represent force hierarchies that characterize patterns using GRL notation
- Include pattern force hierarchy as part of individual pattern templates in the pattern catalog
  - http://jerry.cs.uiuc.edu/~plop/plop2002/final/PatNFR.pdf
- Apply algorithms to match force hierarchies for a given problem domain with known pattern force hierarchies in the pattern catalog
  - Comparison algorithm
    - See “Deciding on a Pattern” by Jonathan C. McPhail and Dwight Deugo
  - Data mining algorithm
  - Learning/Rule-based algorithm
- Perform additional matching on more specific artifacts than force hierarchies as available (e.g., architecture modeling notations, etc.)

Roadmap

Sample EAMF-Driven Pattern Elicitation:

- EAMF Process Patterns (Actors as Agents)
- EAMF GDM UCM Inter-Scenario Relationships
- Enterprise Service Lifecycle
- Service Management Architecture (SMA)

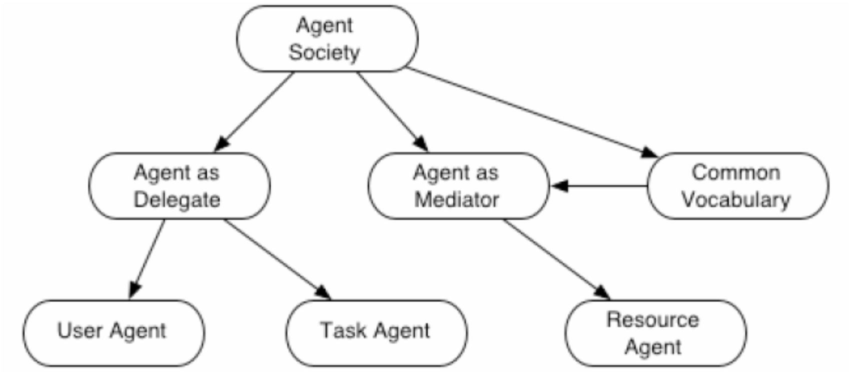
Reasoning About Business Entities and Their Dependencies and Goals



89

Pattern Language Structure for Agent Patterns Selection

(http://www.scs.carleton.ca/~weiss/papers/aois03-revised.pdf)



90

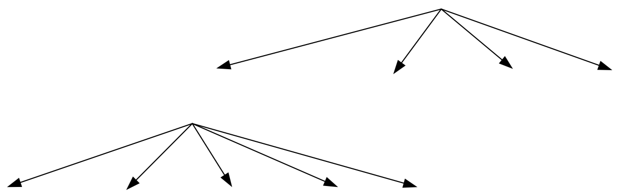
egend:

: GRL M

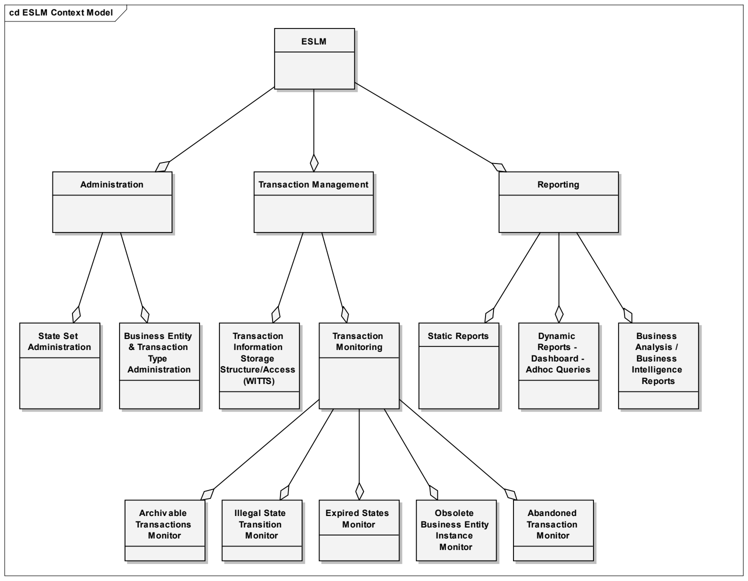
Hi
Business
(bus. o

Inter-Scenario Relationships Design Patterns Used in EAMF's BPM approach

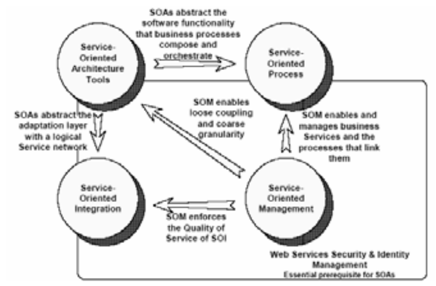
(http://jucmnav.softwareengineering.ca/twiki/pub/UCM/VirLibIsorc2000/isorc2000.pdf, http://www.scs.carleton.ca/~francis/Thesis/phdthesis.pdf)



Enterprise Service Lifecycle Management


cd ESLM Context Model
ESLM
Administration
Transaction Management
Reporting
State Set Administration
Business Entity & Transaction Type Administration
Transaction Information Storage Structure/Access (WITTS)
Transaction Monitoring
Static Reports
Dynamic Reports - Dashboard - Adhoc Queries
Business Analysis / Business Intelligence Reports
Archivable Transactions Monitor
Illegal State Transition Monitor
Expired States Monitor
Obsolete Business Entity Instance Monitor
Abandoned Transaction Monitor

SMA Reference Architecture: The Acronyms…

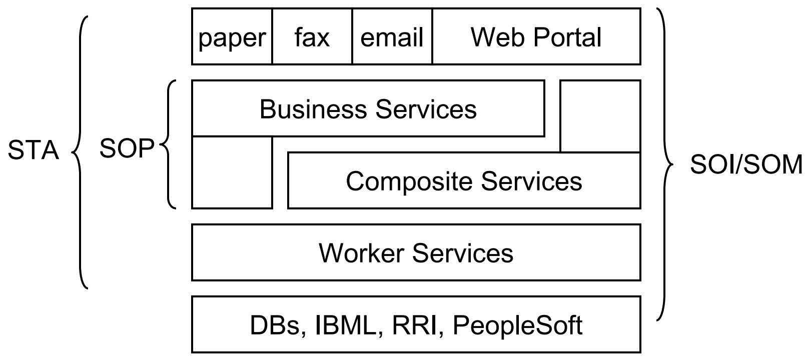


Service-Oriented Architecture Tools
SOAs abstract the software functionality that business processes compose and orchestrate
Service-Oriented Process
SOAs abstract the adaptation layer with a logical Service network
SOM enables loose coupling and coarse granularity
SOM enables and manages business Services and the processes that link them
Service-Oriented Integration
SOM enforces the Quality of Service of SOI
Service-Oriented Management
Web Services Security & Identity Management
Essential prerequisite for SOAs

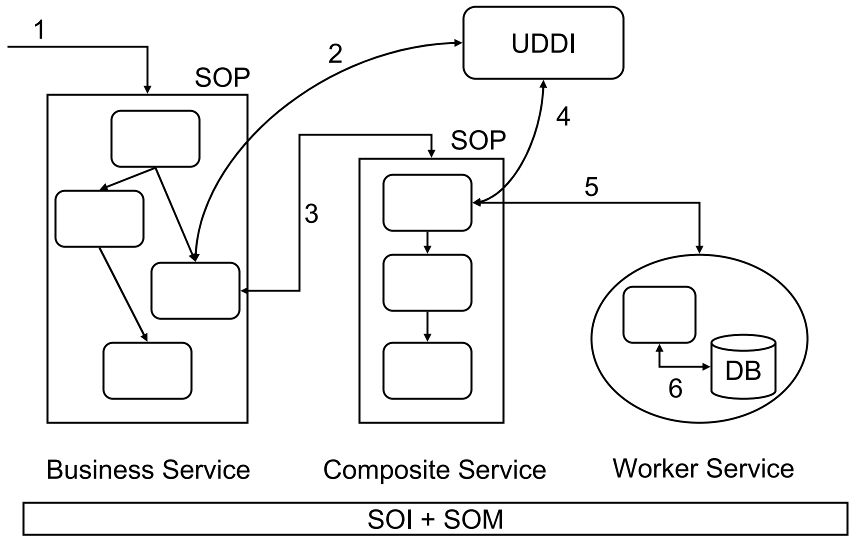
- SMA = SOA + STA
- SOA = SOP + SOI + SOM

- SOA – Service Oriented Architecture
- STA – Service Trader Architecture
- SOP – Service Oriented Process
- SOI – Service Oriented Integration
- SOM – Service Oriented Management

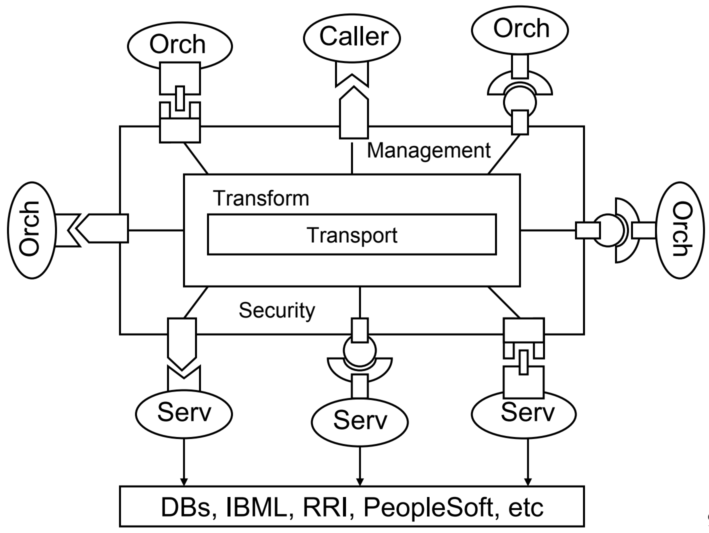
SMA: The Final Big Picture…



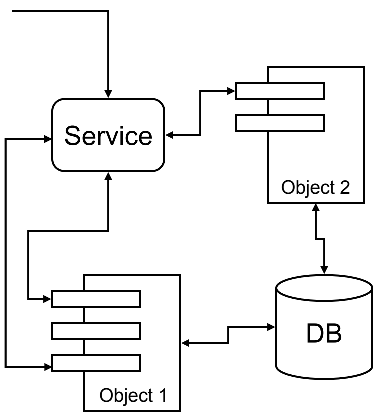
SMA: How it works…



SOI + SOM = ESB
ESB + SOP = SOA

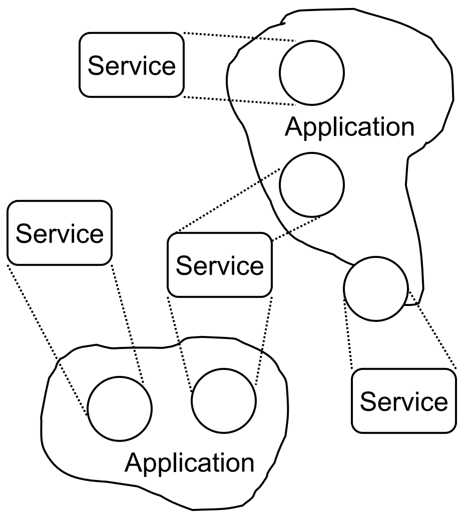


Service Oriented vs Object Oriented



- Services cut across multiple objects
- Objects encapsulate data, but Services encapsulate behavior
- The "lower" the service, the more like an object it becomes

Service Oriented vs Application Oriented



- Applications typically wrap a single "large" database
- "Services" within an application are typically tightly coupled
- Services don't necessarily entirely reside "within" the corresponding application
- Groupings of "high" level services begin to resemble applications

SMA: Technology Properties

- Location and platform independence
- Reuse via shared services (not copied/duplicated code)
- Strict adherence to encapsulation and public API
- Separation of service interface and service implementation (via WSDL)
- Deployment of new services and upgrades to existing services (implementation or interface extension), with NO down-time, via dynamic service end-point discovery (UDDI)

SMA: Long-range Business Benefits

- Uniform and consistent IT deployment "fabric" (virtual platform)
- Uniform and consistent support for SLAs
- Common data security and access control base-line
- Library of common support services that will enable new projects to focus on their unique attributes rather than reinventing infrastructure
- Integrated real-time monitoring and business analysis capabilities across the enterprise
- Enables 24x7 operation, despite software system (service) upgrades

SMA: Benefits, Risks, and Pitfalls

The Good

•Location and platform independence

• Non-proprietary in that ESB enables multiple service "platforms" to interoperate/communicate

• Separation of interface and implementation

• Dynamic late (re)binding

• Live upgrades/(re)deployments

• Flexible reuse supports agile IT support for business

• Enterprise IT architecture/design is driven by business needs

The Bad

• Poor initial API could limit reuse of services (or require cascading redesign/reimplementation)

• Some standards are still emerging or are in flux

• May require "retooling" of system design (thought) process

The Ugly

• More run-time overhead in extra layers and protocols

• Needs coherent and consistent enterprise "platform" management

Roadmap

Classifying Problem Types

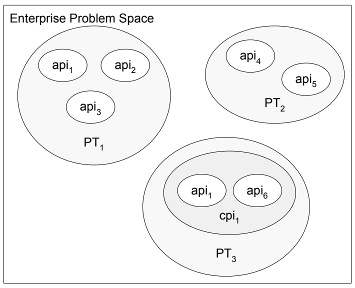
- What is the problem?
- Problem Types
- Enterprise Problem Instance
- Pattern Cluster
- How Does EAMF Come Into Play?

What is the Problem?

- EAMF is a framework with a methodology to capture design information for similar problems
- Before we start designing a solution we need to understand why we are designing a solution
  - In an enterprise, motivations will arise based on needs and/or problems. Change is requested and needs to be dealt with.
  - When we are dealing with change, we need to create a structured environment so that we solve similar problems using similar solutions in order to reduce redundancies
  - This structured environment is not only a technical framework but a structured thinking process which helps us organize the changes in specific categories because if we can categorize the types of changes we can easier categorize the solutions that address them

Problem Types

- **A problem type identifies the essence of a problem without the details of the problem instances**
- **A solution to a problem type is more generic than a solution to a problem instance**
- **By the same token, a generic solution to a problem type can be applied to all problem instances in that problem category**



Problem Types (cont.)

- Atomic problem instance (api)
  - Specific instance of a problem that cannot be further divided into sub-problems
- Composite problem instance (cpi)
  - Specific instance of a problem that can be further subdivided into atomic problem instances, however it would be better to keep this problem instance in its composite form because the solution to the composite problem is more effective than the sum of the solutions to its atomic problem instances
  - In other words, there may be such a synergy between the atomic problem instances which makes the composite problem instance more effectively solvable
- Atomic and composite problem instances are categorized into problem types
  - This creates a smaller set of problem types from a larger set of atomic and composite problem instances

Problem Types (cont.)

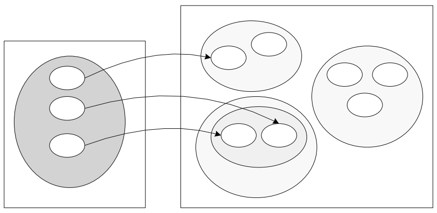
- **Categories of Problem Types**
  - The current problem types need to be determined with the help of business experts and business architect
- **Problem types can be categorized in many different ways**
- **Problem types can be categorized by the type of IT solution they require (IBM does this in their eBusiness) such as**
  - Self-Service
  - Collaboration
  - Information Aggregation
  - Extended Enterprise (B2B)
- **Problem types can be categorized by service line they occur in such as**
  - Billing
  - Claims
  - New Business
  - Etc.

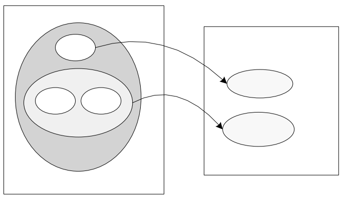
Problem Types (cont.)

- **Categories of Problem Types (cont.)**
  - Problems may be categorized by a more generic classification like
    - Information Sharing
    - Short Lived Business Transaction
    - etc…
  - Yet another way to categorize problems can be the life cycle of a specific context
    - Doculabs consultants did that with their ECM problem types
    - They have categorized the problem types by the life cycle of Content such as
      - Create Content
      - Review Content
      - Approve Content
      - Manage Metadata
      - Etc…
  - We need to find a correct way of categorization in order to achieve our goals.

Enterprise Problem Instance

- Enterprise Problem Instance (epi)
  - Combination of one or more atomic and composite problem instances
- Since these problem instances are categorized as more abstract problem types, enterprise problem instances can be represented as a combination of problem types
- Since these problem instances are categorized as more abstract problem types, enterprise problem instances can be represented as a combination of problem types.





Pattern Cluster

- A **Pattern Cluster** is a set of patterns which are involved in the generic solution of a problem type
- Since there may be more than one generic solution to a problem type:
  - *Enterprise Solution Architectures are a combination of pattern clusters of one or more Enterprise Architectures*
- This conclusion identifies two additional tasks
  - First we need to identify what the available Enterprise Architectures are
  - Second we need to identify the pattern clusters in these Enterprise Architectures.

Pattern Cluster (cont.)

- We have defined Enterprise Architectures as solutions provided within a context
  - In the enterprise landscape not all of the contexts are at the same level of detail
  - It is normal that one Enterprise Architecture uses pieces from another Enterprise Architecture to provide a solution
  - Some of the architectures will be high level and make sense from a business point of view like Customer Relationship Management (CRM)
  - Some of them may be more specific in focus that only deals with certain types of problems like unstructured enterprise information like Enterprise Content Management (ECM)
  - CRM might use some pattern clusters from ECM in order to provide a solution

Pattern Cluster (cont.)

- So far we have established that
  - Specific problem instances are categorized to a problem type
  - A problem type can have one or more enterprise solutions
  - An enterprise architecture contains 1 or more pattern clusters
  - An enterprise solution contains one or more pattern clusters from one or more enterprise architectures

111

How Does EAMF Come Into Play?

- EAMF comes into play from two different views:
  - A catalog view that captures pattern clusters and individual patterns in the framework's catalog matrices
  - A design view that captures enterprise solutions for a problem type and the pattern clusters and individual patterns used in that solution in the framework's standard and domain specific matrices
- The catalog matrices are populated over time
  - Every time a company realizes more experience and knowledge and identifies a pattern or a pattern cluster the cluster would be inserted into the appropriate catalog matrices
  - The catalog matrices would serve as a repository which would help architects and developers to reuse solution structures in future similar problems
  - The standard and domain specific matrices are populated during a project
  - The usage of EAMF begins when a business or technology problem is described. The problem is being analyzed outside the focus of EAMF and a set of business requirements is created based on the problem description and analysis

112

Any Questions?