SERVICE-ORIENTED MODELING FRAMEWORK™ (SOMF™)

VERSION 2.1

SERVICE-ORIENTED SOFTWARE ARCHITECTURE MODEL LANGUAGE SPECIFICATIONS

TABLE OF CONTENTS

INTRODUCTION

ABOUT THE SERVICE-ORIENTED MODELING FRAMEWORK (SOMF)

The service-oriented era has begun. New technologies have emerged to support the "service" notion that signifies, today more than ever, a shift in modern computing whose driving aspects are business imperatives and innovative technological implementations. The service paradigm is not a new concept; however, it emboldens the business perspective of every software development life cycle. Furthermore, unlike the object-oriented approach, which is founded to support modeling of object-based programming languages, the service-oriented modeling framework embodies distinct terminology to foster loose coupling of software assets, reuse of software components, acceleration of time-to-market, reduction of organizational expenditure, and more.

SUPPORTING THE SERVICE-ORIENTED MODELING NOTION

Thus, to support service-oriented modeling activities, SOMF depicts the term "service" as a holistic entity that may encapsulate business requirements, and from a technological perspective, is identified with a software component. This organizational software entity, namely a "service" that is subject to modeling activities, may be any software construct that the enterprise owns, such as an application, software system, system software, Web service, software library, store procedure, database, business process, enterprise service bus, object, cloud computing service, and more.

SO WHAT IS SOMF?

SOMF is a model-driven engineering methodology whose discipline-specific modeling language and best practices focus on software design and distinct architecture activities employed during stages of the software development life cycle. Moreover, architects, analysts, modelers, developers, and managers employ SOMF standalone capabilities or mix them with other industry standard modeling languages to enrich the language syntax, set software development priorities during life cycle stages, and enhance the 360º software implementation view.

SOMF DISCIPLINES AND MODELS

SOMF offers a 360º view of any software development life cycle, starting at the conceptualization phase, supporting design and architecture activities, and extending modeling best practices for service operations in a production environment. To achieve these underpinning milestones, six distinct software development disciplines offer corresponding models whose language notation guide practitioners in designing, architecting, and supporting a service ecosystem:

1. Service-Oriented Conceptualization Model
2. Service-Oriented Discovery and Analysis Model
3. Service-Oriented Business Integration Model
4. Service-Oriented Logical Design Model
5. Service-Oriented Software Architecture Model
6. Cloud Computing Toolbox Model

MODELING GENERATIONS

SOMF diagrams support three chief modeling generations, each of which shows a different time perspective of a software life cycle. These views help practitioners depict business and architectural decisions made at any time during the life span of a software product:

1. *Used-to-Be*. Design and architecture *past state* of a software product and its related environment that were deployed, configured, and operated in production
2. *As-Is*. Design and architecture *current state* of a software product and its corresponding environment that are being operated in production
3. *To-Be*: Design and architecture *future state* of a software product and its associated environment that will be deployed, configured, and operated in production

ABOUT THE SERVICE-ORIENTED SOFTWARE ARCHITECTURE MODEL

This specifications paper focuses on the Service-Oriented Software Architecture Model language whose capabilities and best practices are devised to assist practitioners in planning a loosely coupled and reusable deployment environment. This architectural landscape is populated with technological assets that are distributed across an organization and beyond its boundaries. To fulfill this goal, the language provides a mechanism to abstract and generalize technologies to increase their reuse, develop a common organizational language and technical taxonomy, and promote superior software asset collaboration and integration. Hence, the Software Architecture Model offers three distinct notations (refer to the Notation Section for further details):

1. Reference Architecture
2. Conceptual Architecture
3. Logical Architecture

Consider the chief benefits of the Service-Oriented Software Architecture Model language:

- Generalizing architectural concepts by employing architectural metaphors
- Providing technological direction
- Establishing a reference architecture for projects and larger development initiatives
- Offering best practices and guidance to drive the adoption of innovative technologies
- Aligning architecture initiatives with business goals
- Establishing business ownership
- Developing a technology stack for an architecture initiative
- Encouraging software reuse
- Fostering software asset consolidation
- Alleviating interoperability challenges
- Focusing on software asset deployment, consumption, and utilization

NOTATION SECTION

REFERENCE ARCHITECTURE

Reference architecture is a template of common technologies, software components, and applications that offer a solution to an organizational concern. Architects, analysts, developers, and managers can employ this template to drive business and technical development of software. Furthermore, reference architecture should also offer best practices and provide a list of selected technologies that should be employed during development and deployment of software to a production environment. To fulfill this goal, a Reference Architecture diagram should be developed to communicate an architecture model to the business and development community.

REFERENCE ARCHITECTURE DIAGRAM BUILDING BLOCKS

As discussed, a Reference Architecture diagram offers a solution to organizational problems. This diagram should depict the chief building blocks of an architecture model to drive business and technological initiatives in the enterprise. Therefore, the practitioner should focus on constructing at least two types of architecture templates: Conceptual Reference Architecture and Logical Reference Architecture. Figure 1 illustrates the chief building blocks of a reference architecture that can be used to compose either a conceptual or a logical diagram.

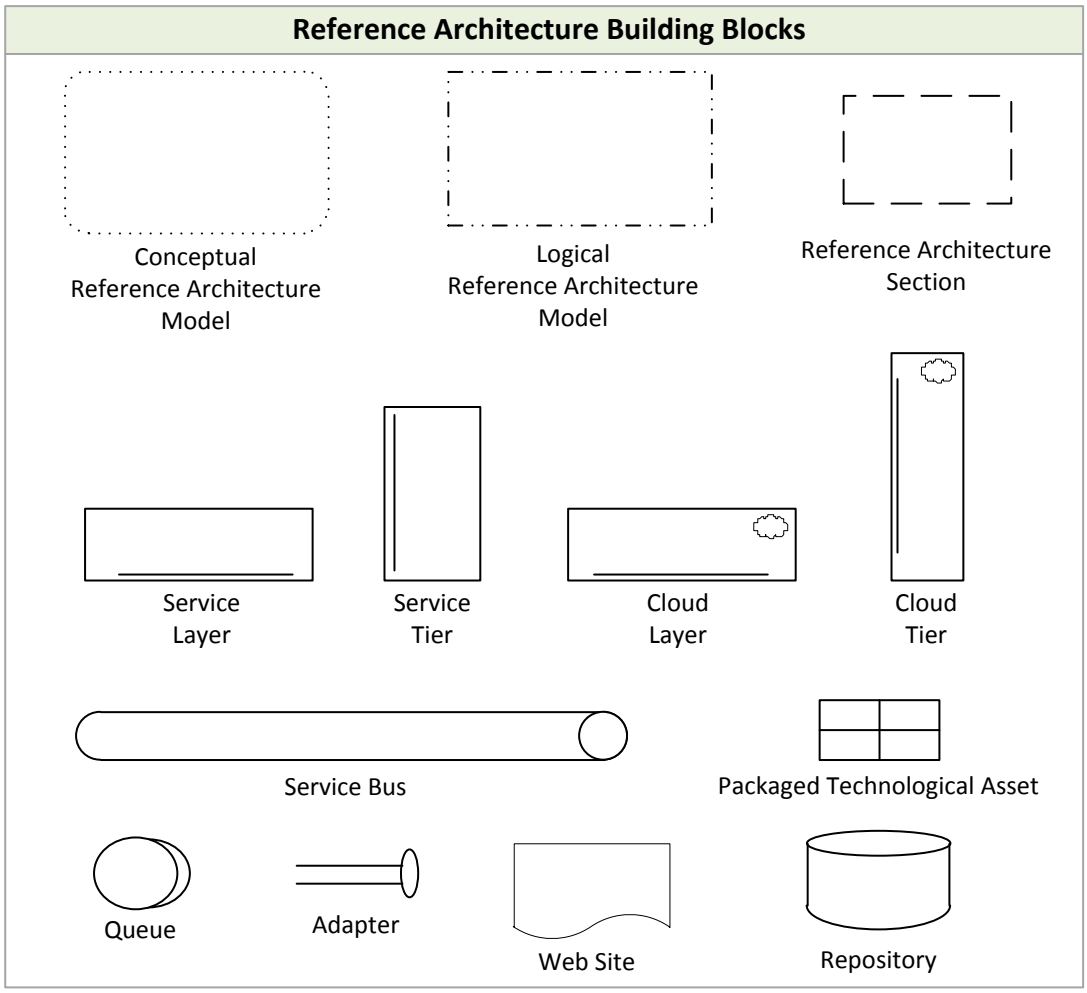

FIGURE 1: REFERENCE ARCHITECTURE BUILDING BLOCKS

- *Conceptual Reference Architecture Model.* A boundary that contains the conceptual components of a Reference Architecture diagram
- *Logical Reference Architecture Model.* A boundary that contains the logical components of a Reference Architecture diagram
- *Reference Architecture Section.* A section within a conceptual or logical Reference Architecture diagram. This section may be used to group concepts or technologies, such as a collection of repositories, group of applications, group of services, etc.
- *Service Layer.* A horizontally positioned Reference Architecture diagram area whose contained services depend on the diagram building block positioned beneath. This dependency may pertain to functionality, processes, technological concepts, logical affiliation, product capabilities, deployment and configuration aspects, and more
- *Service Tier.* A vertically positioned Reference Architecture diagram area whose contained services contribute to the adjacent diagram's building block positioned vertically or horizontally. This contribution may pertain to functionality, processes, technological concepts, logical affiliation, product capabilities, deployment and configuration aspects, and more
- *Cloud Layer.* A horizontally positioned Reference Architecture diagram area whose contained cloud services depend on the diagram building block positioned beneath. This dependency may pertain to functionality, processes, technological concepts, logical affiliation, product capabilities, deployment and configuration aspects, and more

- *Cloud Tier.* A vertically positioned Reference Architecture diagram area whose contained cloud services contribute to the adjacent diagram's building block positioned vertically or horizontally. This contribution may pertain to functionality, processes, technological concepts, logical affiliation, product capabilities, deployment and configuration aspects, and more
- *Service Bus.* A Reference Architecture diagram component that represents an enterprise message bus (ESB)
- *Packaged Technological Asset.* Any software entities bundle that is ready to be deployed or is already installed in a production environment. This bundle of software executables may contain services, application servers, applications, off-the-shelf products, software components, and more
- *Repository.* Represents any type of information storage facility, such as a database, meta data repository, and more
- *Queue.* A message bus component that enables asynchronous communication by storing messages posted by the message sender until retrieved by the message receiver
- *Adapter.* A software component that is installed at an application site to provide interfaces and widen services to outside consumers
- *Website.* A computer or an application server connected to the Internet or intranet whose Web pages and digital media content such as images and videos offer services to external or internal consumers

REFERENCE ARCHITECTURE MODEL CONNECTORS

Use the reference architecture connectors illustrated in Figure 2 to link the building blocks discussed in the Reference Architecture Diagram Building Blocks Section. As is apparent, there are two sets of connectors: Concrete Message Exchange Dependency and Abstract Message Exchange Dependency. For constructing a logical Reference Architecture diagram, use the former set; the latter should be employed when building a conceptual Reference Architecture diagram. Combine the two when necessary.

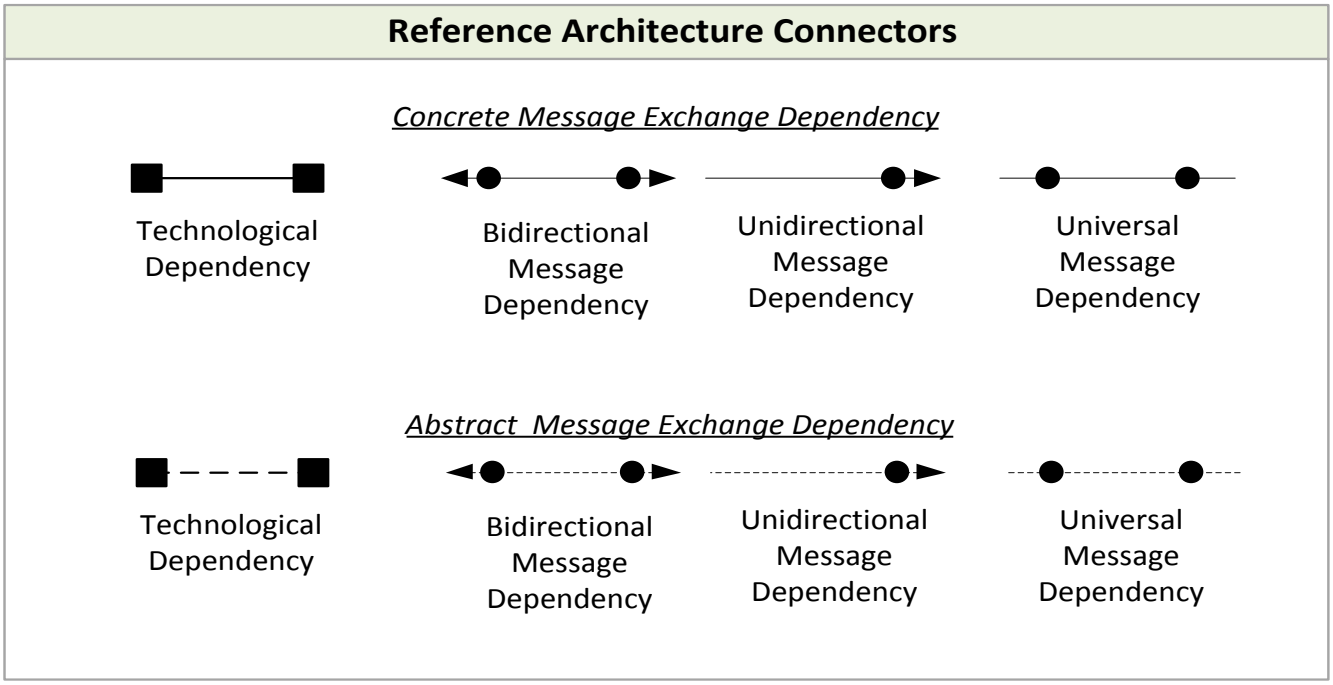

FIGURE 2: REFERENCE ARCHITECTURE MODEL CONNECTORS

- *Technological Dependency.* Identifies reliance between two reference architecture building blocks. This dependency may pertain to general dependency of products in terms of configuration aspects, deployment, design and architecture, and more
- *Bidirectional Message Dependency.* Indicates a two-way message exchange between two reference architecture building blocks

- *Unidirectional Message Dependency.* Indicates one-way message routing between two reference architecture building blocks
- *Universal Message Dependency.* Signifies a general message dependency that does not identify any message routing direction. This dependency may be replaced with a more specific message path direction once the architecture has been finalized

CLOUD TYPING TAGS

If a project or an architecture initiative involves cloud computing modeling activities, any individual cloud layer or tier may require typing. The term "typing" pertains to cloud categorization to help understand the design model that is applied to a production environment. Tagging a cloud by the proper tag (illustrated in Figure 3) would also indicate the types of consumers allowed to utilize a cloud facility and its offered services.

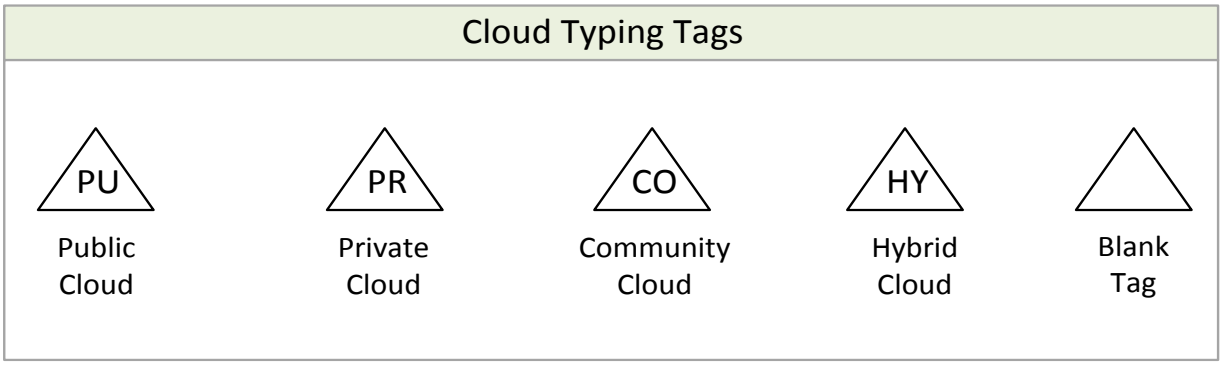

FIGURE 3: CLOUD TYPING TAGS

- *Public Cloud Tag.* Identifies a cloud that is maintained by an off-site party service provider that offers configurable features and deployments charged to subscribed Internet consumers
- *Private Cloud Tag:* Indicates a cloud of services that is sponsored, maintained, and operated by an organization, available only on private networks, and utilized exclusively by internal consumers
- *Community Cloud Tag.* Identifies a cloud whose services are consumed by two or more organizations that share similar business or technical requirements
- *Hybrid Cloud Tag.* Depicts a cloud that combines the properties of two or more cloud types described on this list
- *Blank Tag.* Enables other cloud definitions that are not a part of this list

CONCEPTUAL ARCHITECTURE

A conceptual architecture illustrates an environment whose technological components are generalized by emphasizing capabilities of software components to provide a solution from a functionality perspective, rather than describing their concrete deployment and configuration aspects. Moreover, a Conceptual Architecture diagram may be used to communicate a technology stack, layers of software dependencies that collaborate to provide a solution. Practitioners who typically communicate a direction for software development can also link concepts to each other to develop a conceptual architecture environment.

CONCEPTUAL ARCHITECTURE ASSETS

Use the assets illustrated in Figure 4 to construct a Conceptual Architecture diagram. Refer to the Examples Section to understand the employment of these conceptual architecture assets and their contribution to developing simplified operating environments.

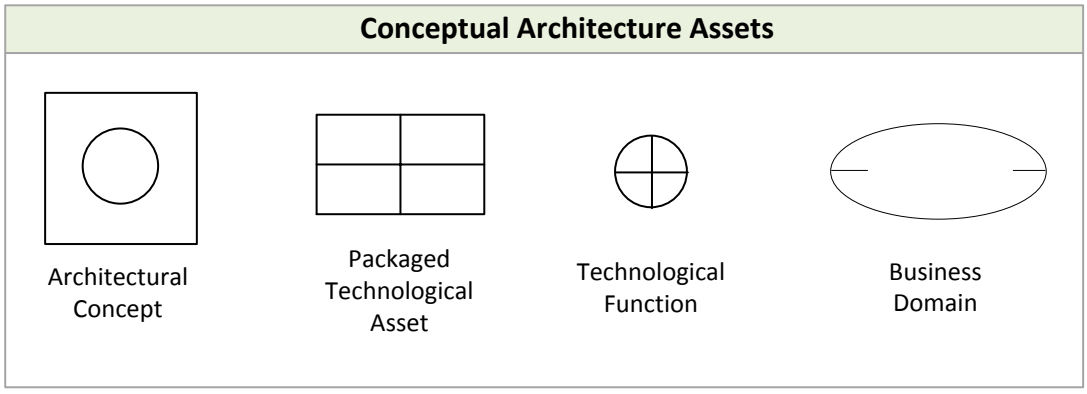

FIGURE 4: CONCEPTUAL ARCHITECTURE ASSETS

- *Architectural Concept.* A generalized technological asset whose functionality contributes to a business or technical solution. For example, an architecture concept may be a "Service Hub," "Virtual Machine," "Insurance Claims Processor," "Business Rules Engine," etc.

- *Packaged Technological Asset.* Any software entities bundle that contains technical products, such as services, application servers, applications, off-the-shelf products, software components, and more
- *Technological Function.* Described capability of a technical asset in terms of functionality and capacity to provide a partial or whole solution. For example, "Rule Based Message Routing," "Message Transformation," or "Message Orchestration"
- *Business Domain.* An identified business owner, sponsor, line of business, or business organization that supports applications, services, or any technological asset

CONCEPTUAL ARCHITECTURE CONNECTORS

To link the conceptual architecture assets that are described in the previous section, the conceptual architecture connectors illustrated in Figure 5 should be employed. Practitioners should use these connectors to affiliate architectural concepts, build a technology stack, abstract a particular technology to increase software reuse, or depict a technical asset's capabilities.

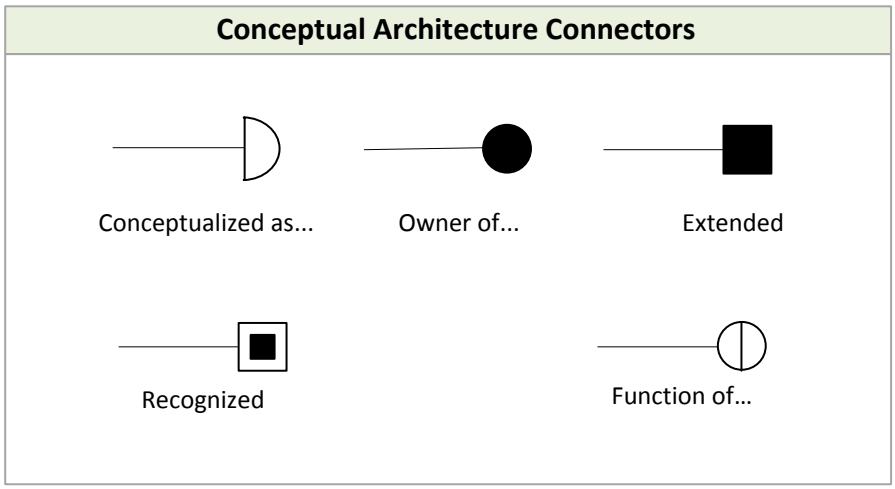

FIGURE 5: CONCEPTUAL ARCHITECTURE CONNECTORS

- *Conceptualized as….* Derives an architectural concept that is generalized from a technological asset. Identifies an abstraction that describes a technological idea
- *Owner of…..* Identifies ownership of business or technology organizations that sponsor and are accountable for the operations of a technological asset
- *Extended.* Denotes a technology stack and the dependencies of its components on each other
- *Recognized.* Describes potential collaboration and affiliation between two architectural concepts

- *Function of….* Identifies a technological asset's functionality, features, and capabilities to contribute to a solution

LOGICAL ARCHITECTURE

A logical architecture is about connecting the dots before deploying a packaged technical asset to a production environment. The term "connecting the dots" refers to identifying an integration scheme, and planning concrete links and message paths that should be used to pass information and execute transactions between packaged technological assets in a production environment.

In addition, a Logical Architecture diagram reflects the technological assets' behavioral patterns and execution of business processes. The diagram also identifies reusability opportunities and reduction of asset dependencies. By doing so, an organization can avoid tightly coupled implementations and control unwieldy software asset distribution conditions.

LOGICAL ARCHITECTURE ASSETS AND CONNECTORS

Use the logical architecture assets and connectors illustrated in Figure 6 to construct a Logical Architecture diagram. These simple notations should be employed to focus on packaged technological assets' functionality and message utilization of services, applications, off-the-shelf products, and more.

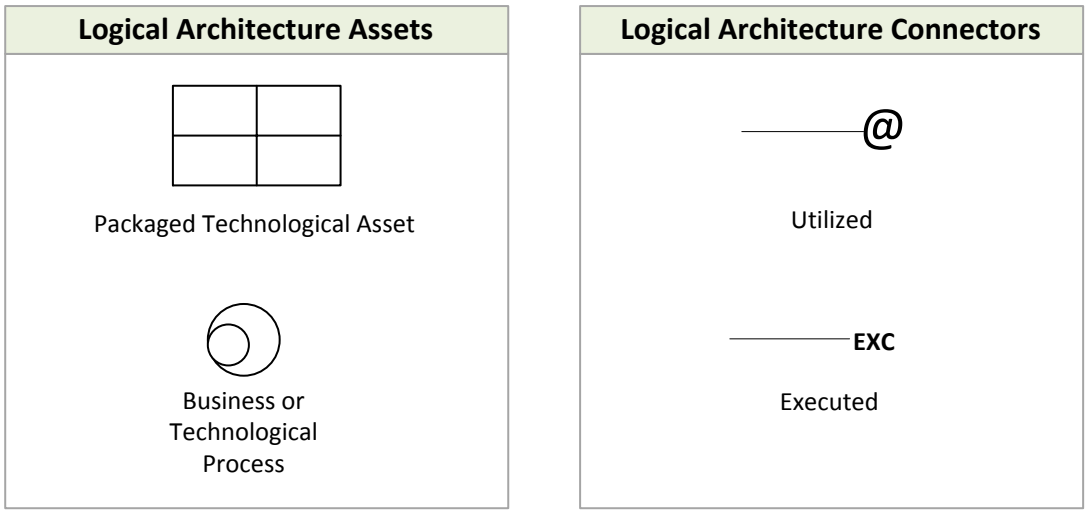

FIGURE 6: LOGICAL ARCHITECTURE ASSETS AND CONNECTORS

Logical Architecture Assets

- *Packaged Technological Asset.* Any software entities bundle that contains technical products, such as Web services, application servers, applications, off-the-shelf products, software components, and more
- *Business or Technological Process.* Denotes a process of a deployed and packaged technological asset

Logical Architecture Connectors

- *Utilized.* Identifies utilization, consumption, dependencies, collaboration, and integration aspects of the packaged technological entities participating in an Asset Utilization diagram (refer to the Examples Section to view this diagram)
- *Executed.* Denotes a business process that takes part in a logical architecture solution executed by the various packaged technological assets. This symbol is used in the Transaction Directory diagram (refer to the Examples Section to view this diagram)

MODELING SPACES

A modeling space (illustrated in Figure 7) is a defined area in which conceptual architecture or logical architecture modeling activities take place. This area also identifies boundaries of organizations, and containment scope of services, service clusters, or cloud computing environments.

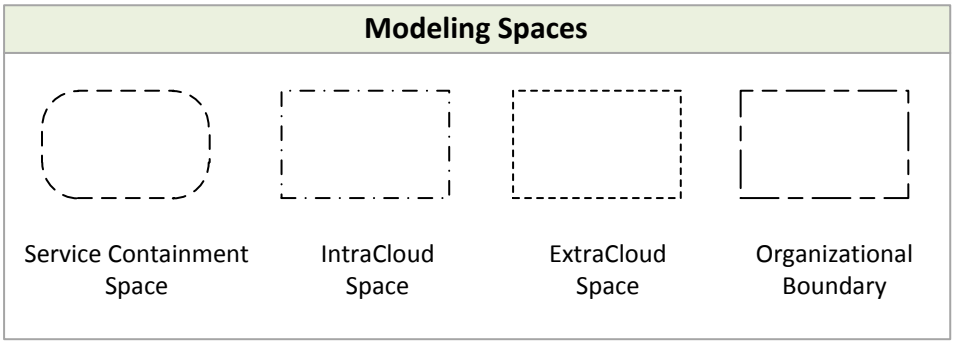

FIGURE 7: MODELING SPACES

- *Service Containment Space.* An area that identifies the aggregated child services contained in a parent composite service or service cluster. This space can also define any collaboration of a service groups that are gathered to offer a solution
- *IntraCloud Space.* A modeling area that shows services that operate in a cloud
- *ExtraCloud Space.* A modeling area that depicts services that operate outside of a cloud
- *Organizational Boundary.* A computing area of an organization, such as a division, department, company, partner company, consumer, or community

EXAMPLES SECTION

REFERENCE ARCHITECTURE DIAGRAM

There are two types of architecture diagrams: logical and conceptual. These diagrams communicate a technological direction, providing a template for software product integration, dependency, and utilization best practices. The Logical Reference Architecture diagram identifies message dependency and concrete correlations between deployment layers and tiers. A logical reference architecture may also include specific applications, services, and off-the-shelf products. Conversely, the conceptual architecture is concerned more about relationships of architectural concepts and generalization of an architectural solution. In most cases, a conceptual reference architecture does not refer to specific low-level technologies.

LOGICAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 8)

a. Reference architecture model: Logical
b. Service tiers: Service Development Platform, Management Services
c. Service layers: Business Services, Business Integration Services, Data Collection Services, Data Access Services
d. Other building blocks: ESB, Repository Section
e. Connectors: Concrete Bidirectional Message Dependency, Concrete Universal Message Dependency

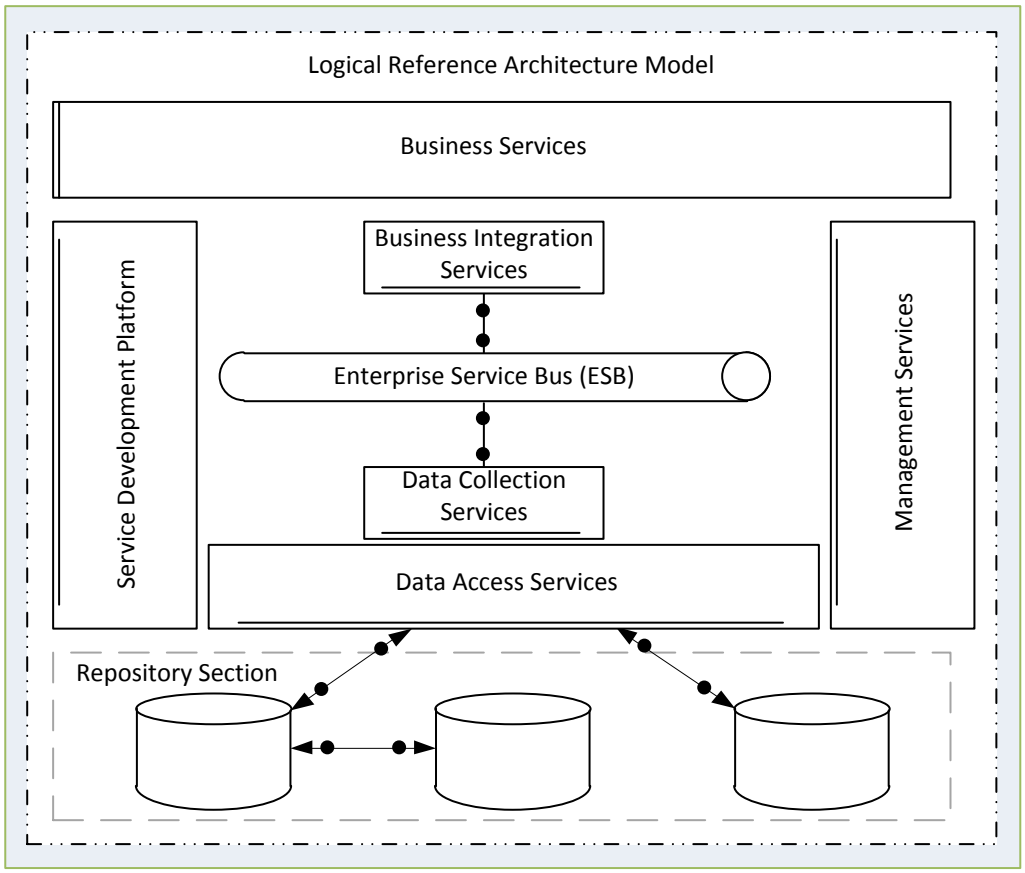

FIGURE 8: LOGICAL REFERENCE ARCHITECTURE DIAGRAM
USING SERVICE TIERS, LAYERS, ESB, AND A REPOSITORY SECTION

LOGICAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 9)

a. Reference architecture model: Logical
b. Service layer: Business Applications
c. Cloud layer: Cloud Layer – SaaS tagged as "PU" (Public)
d. Cloud tier: Cloud Tier – IaaS tagged as "PU" (Public)
e. Other building blocks: Enterprise Service Bus (ESB)

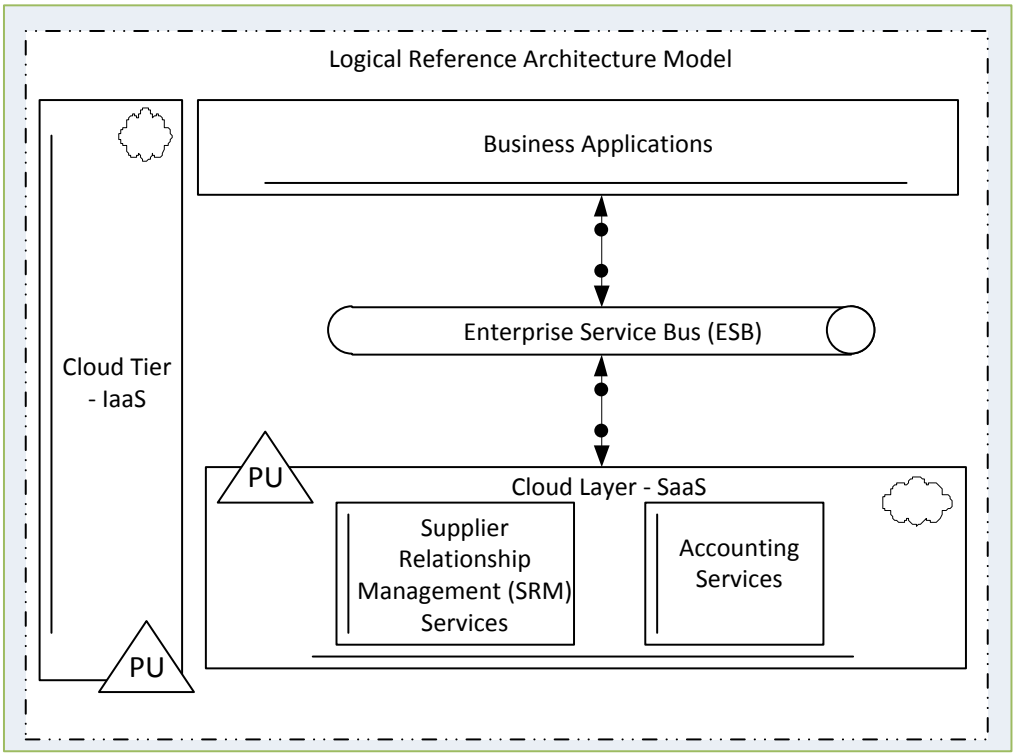

FIGURE 9: LOGICAL REFERENCE ARCHITECTURE DIAGRAM
WITH CLOUD TIER, CLOUD LAYER, ESB,
AND BUSINESS APPLICATIONS SERVICE LAYER

LOGICAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 10)

a. Reference architecture model: Logical
b. Service layers: Business Rules Engine Components, Orchestration Engine Components, Insurance Underwriting Engine Components
c. Service tier: Application Server Components
d. Connectors: Concrete Bidirectional Message Dependency, Concrete Unidirectional Message Dependency

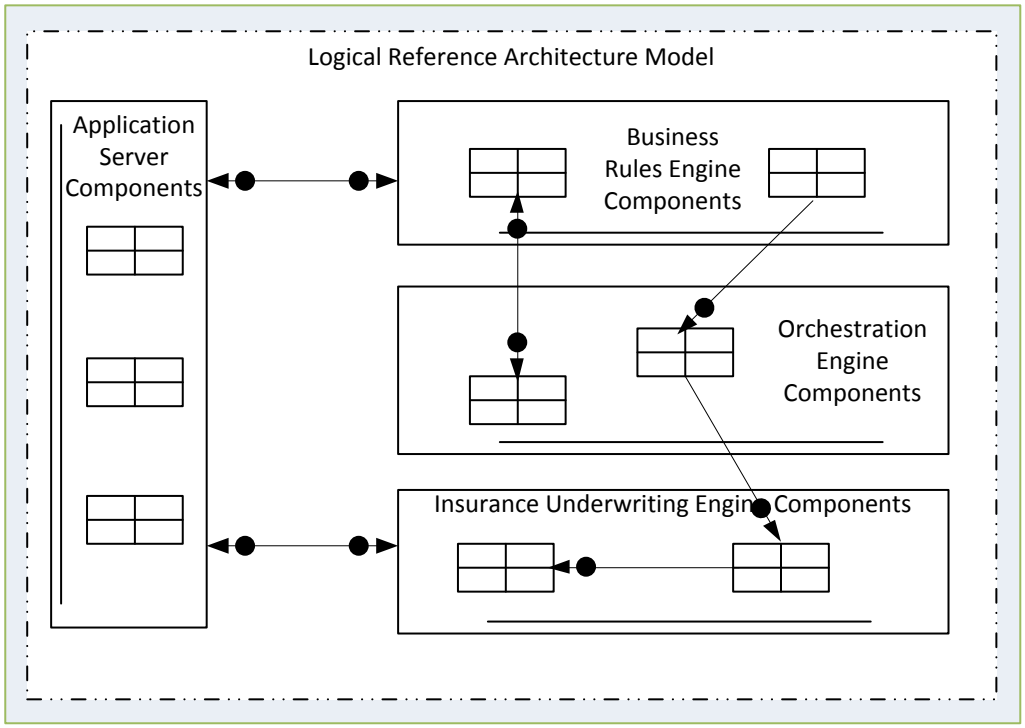

FIGURE 10: A LOGICAL REFERENCE ARCHITECTURE WITH SERVICE TIER AND LAYERS USING CONCRETE BIDIRECTIONAL AND UNIDIRECTIONAL MESSAGE DEPENDENCY CONNECTORS

CONCEPTUAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 11)

a. Reference architecture model: Conceptual
b. Service layers: Business News Services, World News Locator, Local News Services
c. Service tier: Data Aggregation Services

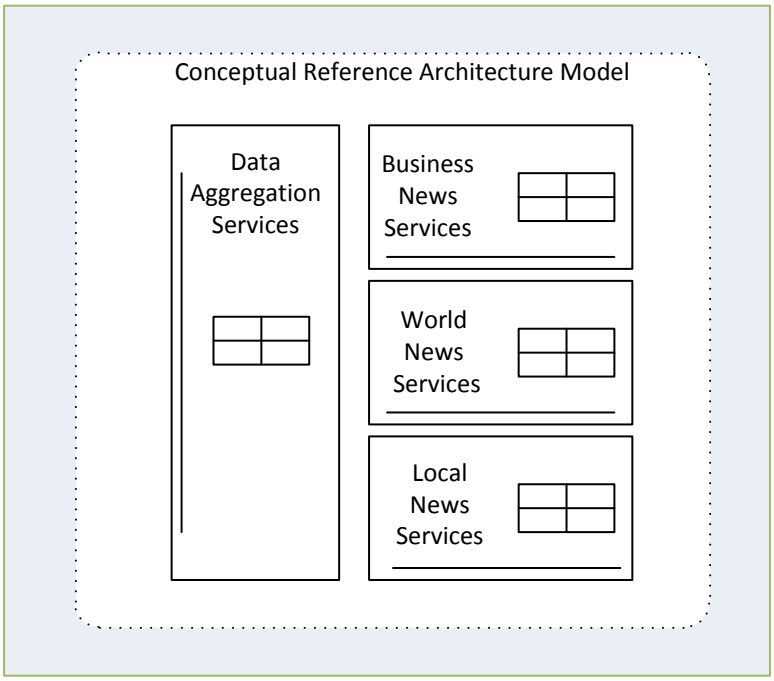

FIGURE 11: A GENERIC CONCEPTUAL ARCHITECTURE MODEL
WITH INHERIT LAYERS AND TIER DEPENDENCIES
WITHOUT CONNECTORS

CONCEPTUAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 12)

a. Reference architecture model: Conceptual
b. Cloud tier: Hybrid (HY) Vendor Information Cloud
c. Hybrid Vendor Information Cloud contains:
    i. Name and Address Services
    ii. Document Processing Services
d. Service layers: Accounting Services, Check Printing Services, Accounts Payable Services
e. Connectors: Abstract Technological Dependency

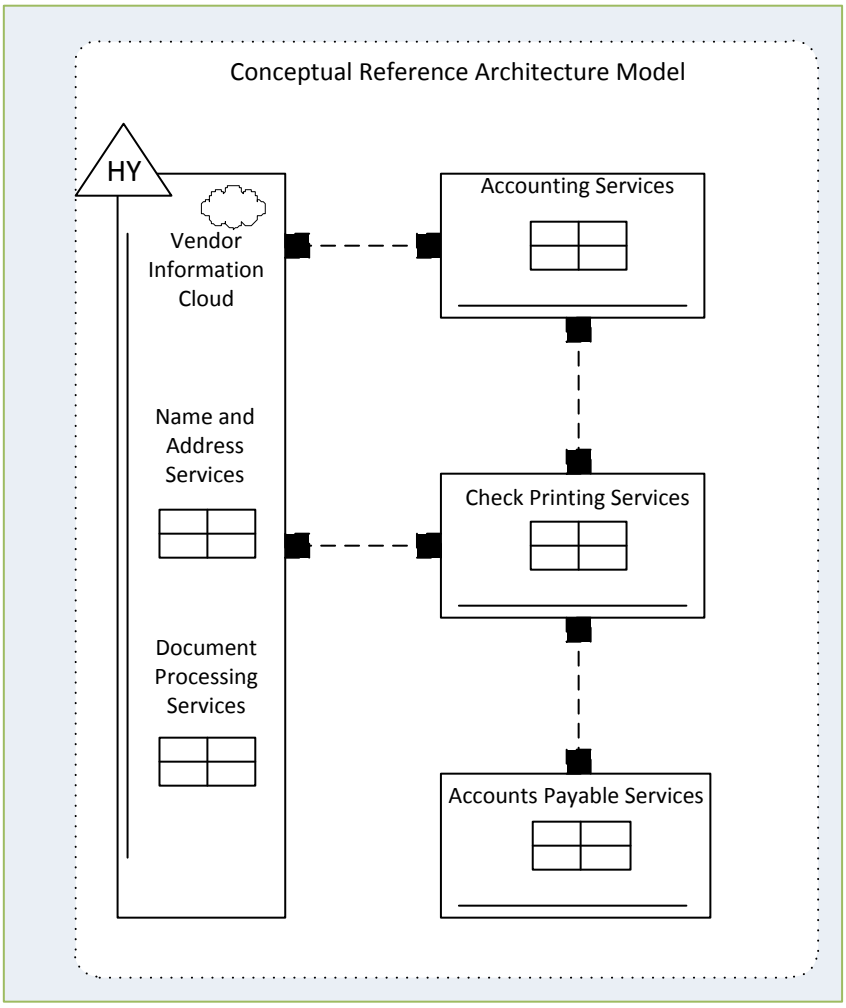

FIGURE 12: CONCEPTUAL ARCHITECTURE MODEL
WITH CLOUD OF SERVICES TIER
AND LAYERS OF SERVICE DEPENDENCIES

CONCEPTUAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 13)

a. Reference architecture model: Conceptual
b. Service layers: Business Application, Application Server, Operating System
c. Connectors: Extended

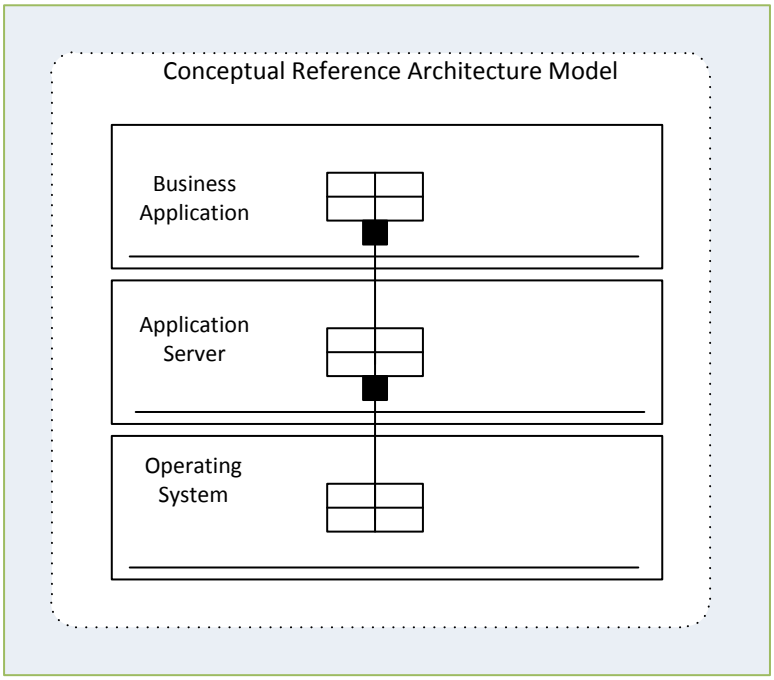

FIGURE 13: CONCEPTUAL REFERENCE ARCHITECTURE MODEL
USING THE EXTENDED CONNECTOR
TO DEPICT A TECHNOLOGY STACK

CONCEPTUAL REFERENCE ARCHITECTURE DIAGRAM (FIGURE 14)

a. Reference architecture model: Conceptual
b. Service layers: Conceptual Services, Credit Verification Web Services, Loan Administration Line of Business,
c. Connectors: Owner of…, Conceptualized as…, Recognized

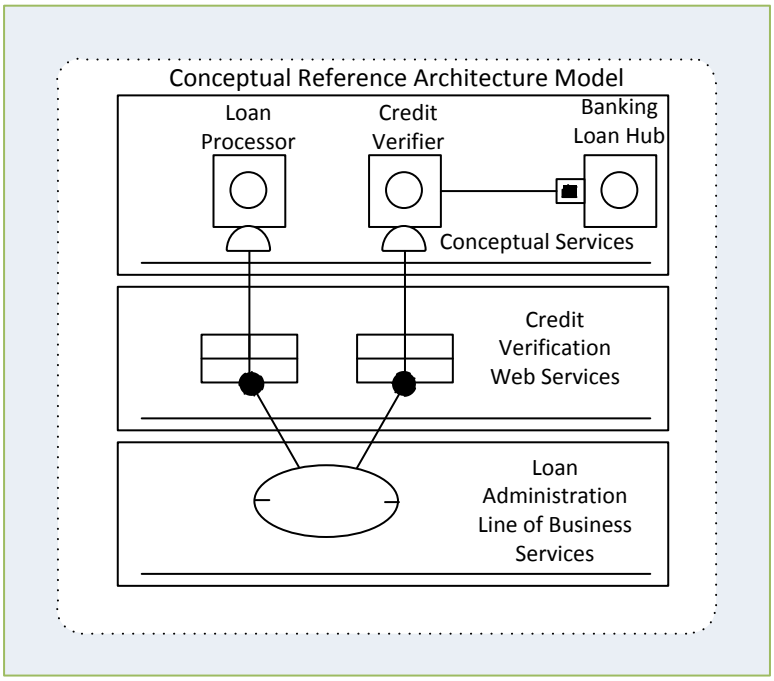

FIGURE 14: CONCEPTUAL REFERENCE ARCHITECTURE MODEL USING CONCEPTUAL ARCHITECTURE CONNECTORS

CONCEPTUAL ARCHITECTURE DIAGRAM

A Conceptual Architecture diagram communicates technical commonalities and generalization of technologies. The used notation identifies metaphors of software implementations to foster a common architecture language and vocabulary. There are typically three layers of abstractions demonstrated in a conceptual architecture artifact, as illustrated in example depicted in Figure 15: Architectural Concepts that are derived from a Technological Environment whose Business Environment is typically affiliated with sponsorship and financial support. Note, not all layers are required in a Conceptual Architecture diagram.

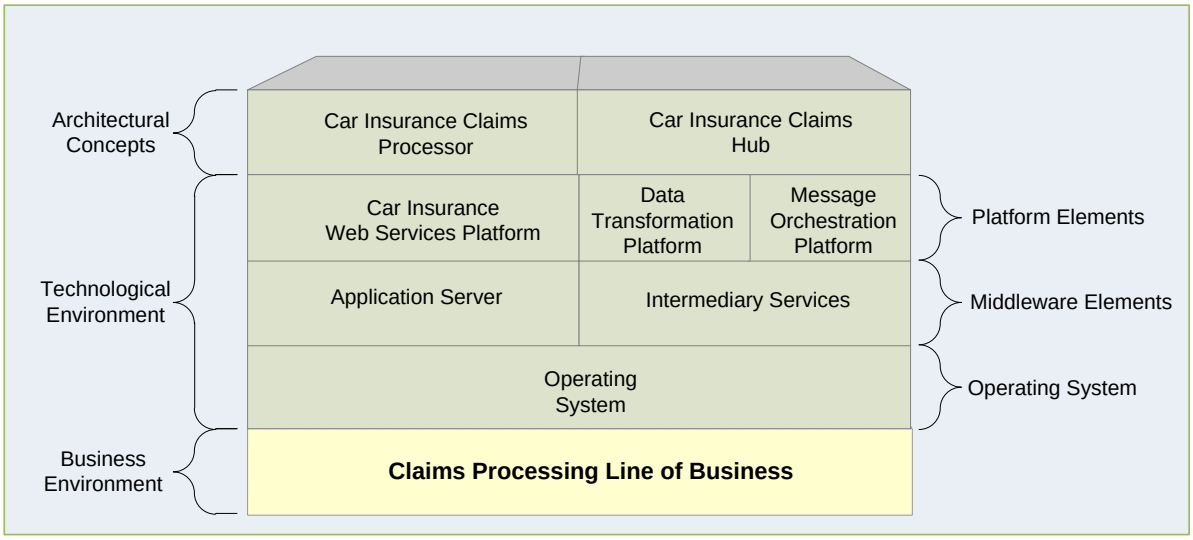

FIGURE 15: THREE LAYERS OF A CONCEPTUAL ARCHITECTURE DIAGRAM EXAMPLE

CONCEPTUAL ARCHITECTURE DIAGRAM (FIGURE 16)

a. Architecture concept: Insurance Claims Hub
b. Packaged technological assets: Message Routing Web Services, Data Transformation Web Services
c. Business domain: Car Insurance Line of Business
d. Connector: Conceptualized as… , Owner of…

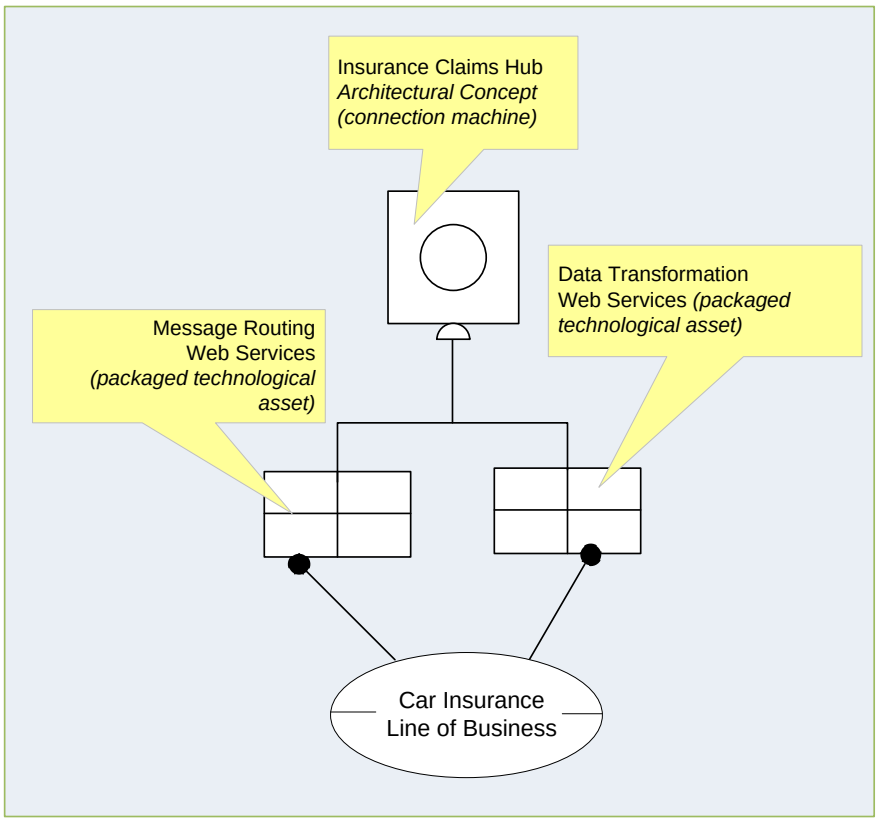

FIGURE 16: CONCEPTUAL ARCHITECTURE DIAGRAM WITH THREE LAYERS, 1) AN ARCHITECTURAL CONCEPT DERIVED FROM 2) TECHNOLOGICAL ASSETS OWNED BY A 3) BUSINESS DOMAIN

CONCEPTUAL ARCHITECTURE DIAGRAM (FIGURE 17)

a. Architecture concept: Trading Transaction Workflow, Trading Hub, Trading Archived
b. Connectors: Recognized

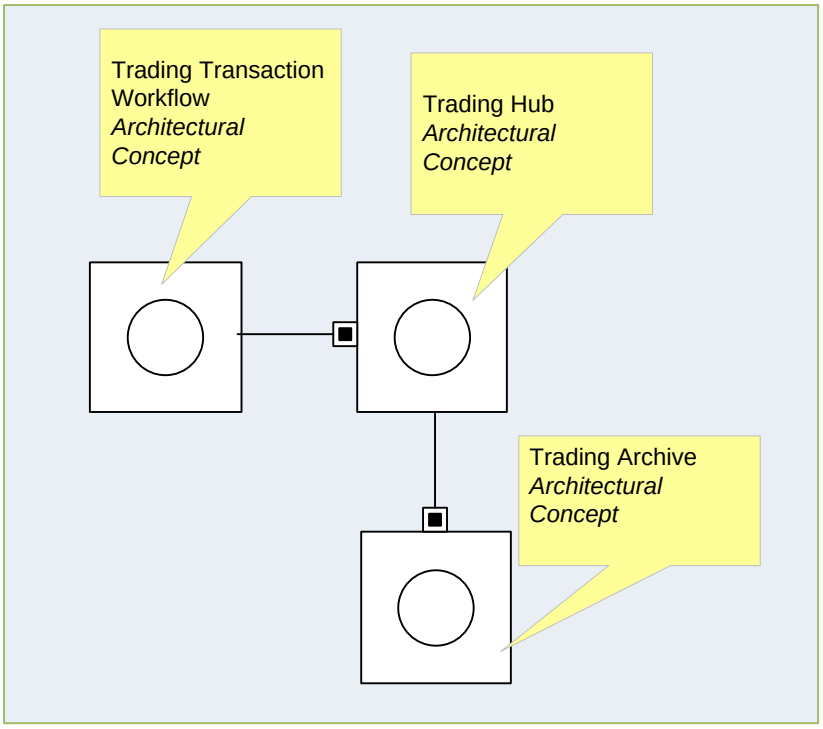

FIGURE 17: CONCEPTUAL ARCHITECTURE DIAGRAM WITH REUSABLE ARCHITECTURAL CONCEPTS

a. Hybrid IntraCoud Space Trading Repositories contains: Trading Archive Architectural Concept
b. ExtraCloud Space contains: Trading Transaction Workflow Architectural Concept, Trading Hub Architectural Concept
c. Connectors: Recognized

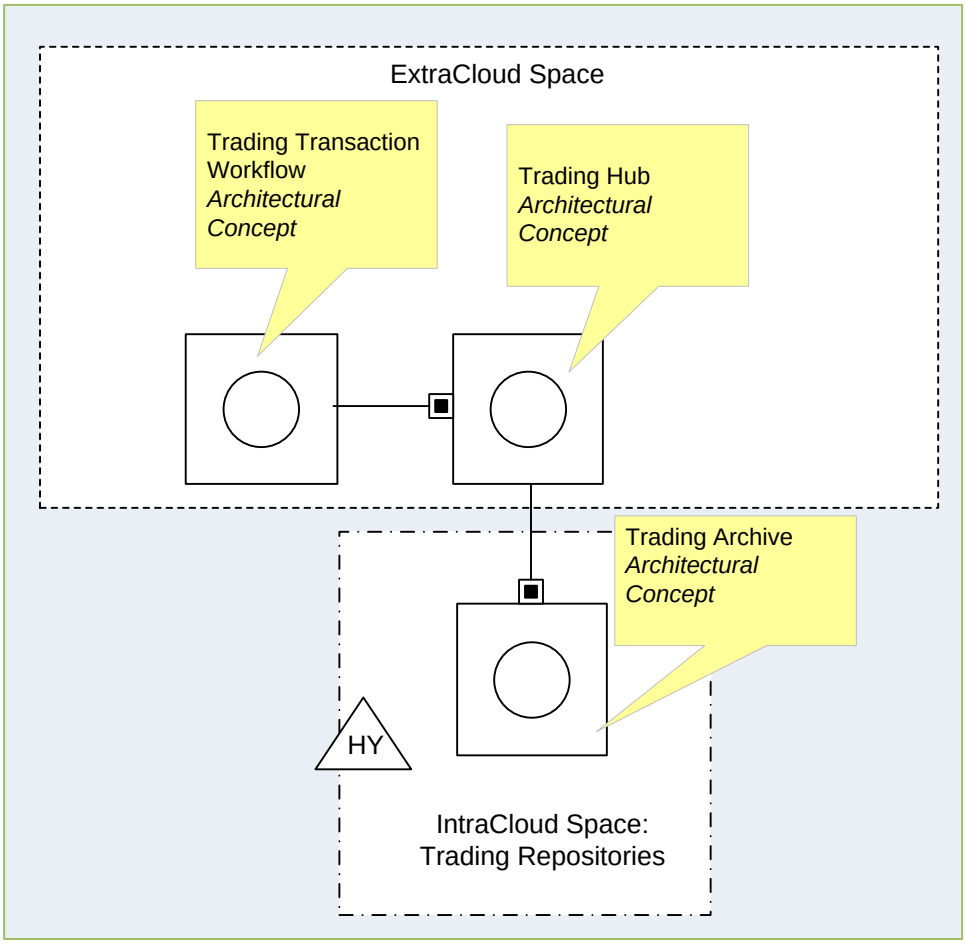

FIGURE 18: CONCEPTUAL ARCHITECTURE DIAGRAM
WITH REUSABLE ARCHITECTURAL CONCEPTS
ACROSS INTRACLOUD SPACE AND EXTRACLOUD SPACE

CONCEPTUAL ARCHITECTURE DIAGRAM (FIGURE 19)

a. Technological asset: ESB
b. Technological functions: Rule Based Message Routing, Protocol Bridging, Message Transformation
c. Connectors: Function of…

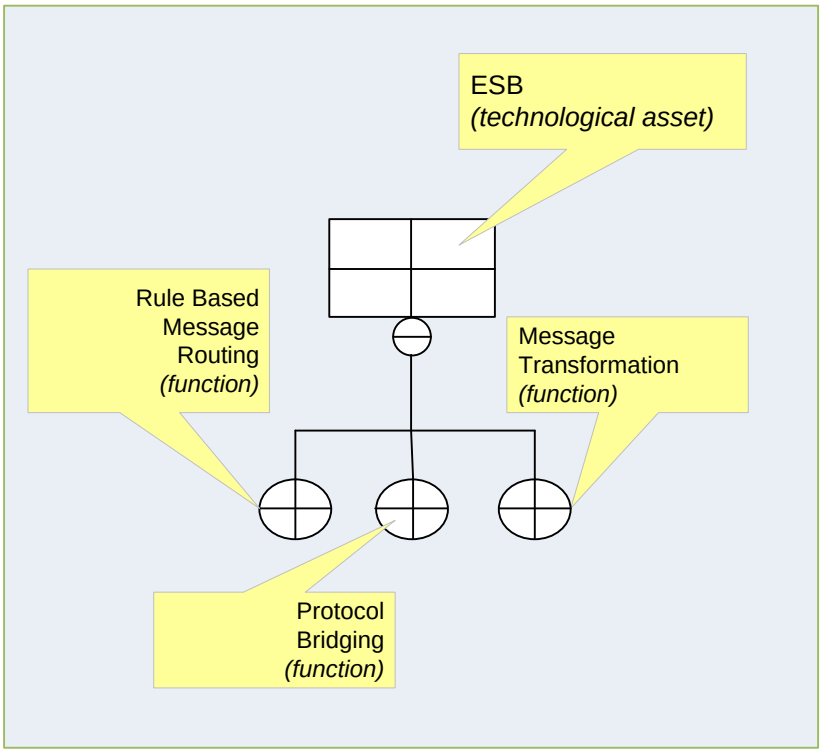

FIGURE 19: DEPICTION OF AN ESB TECHNOLOGICAL CAPABILITIES

LOGICAL ARCHITECTURE DIAGRAMS

A logical architecture environment should be viewed as a technology pyramid, as depicted in Figure 20. The three logical layers that make up a logical environment are the foundation of the Asset Utilization diagram: Technology Foundation, Enabling Technologies, and Reusable Software. The Transaction Directory diagram is another perspective of the logical architecture, by which processes are cataloged to communicate the functionality of technological assets. Use these diagrams to elaborate on integration, collaboration, and deployment of an architectural environment. Connecting the "dots" should be driving this initiative.

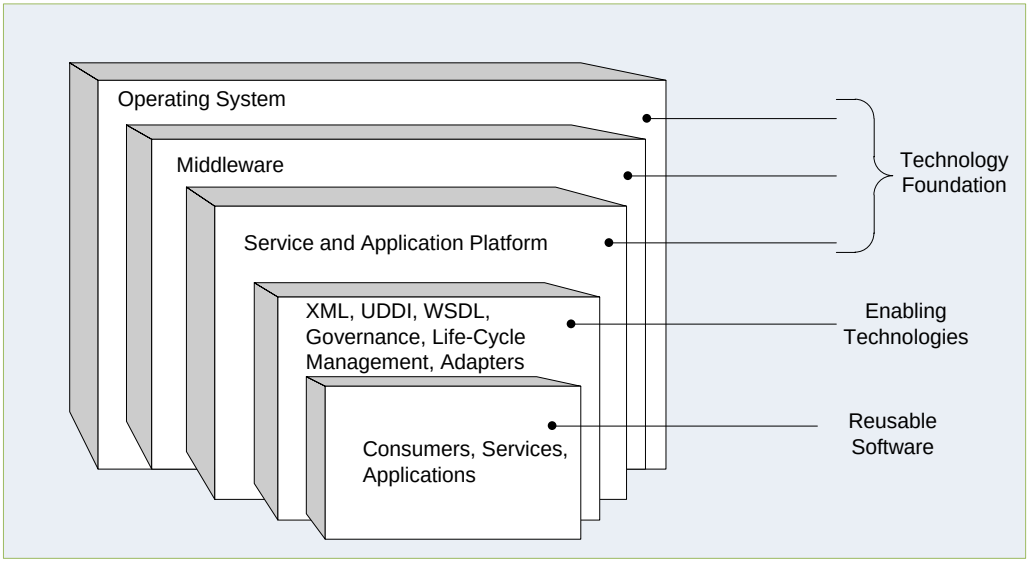

FIGURE 20: LOGICAL ARCHITECTURE TECHNOLOGY PYRAMID

ASSET UTILIZATION DIAGRAM (FIGURE 21)

a. Mainframe Platform contains: General Ledger Services, Bank Reconciliation Services, Reporting Services
b. Distributed J2EE Platform contains two tiers: Accounts Payable Web Services in the Accounts Payable Tier, Accounts Receivable Web Services in the Accounts Receivable Tier
c. Connectors: Utilized

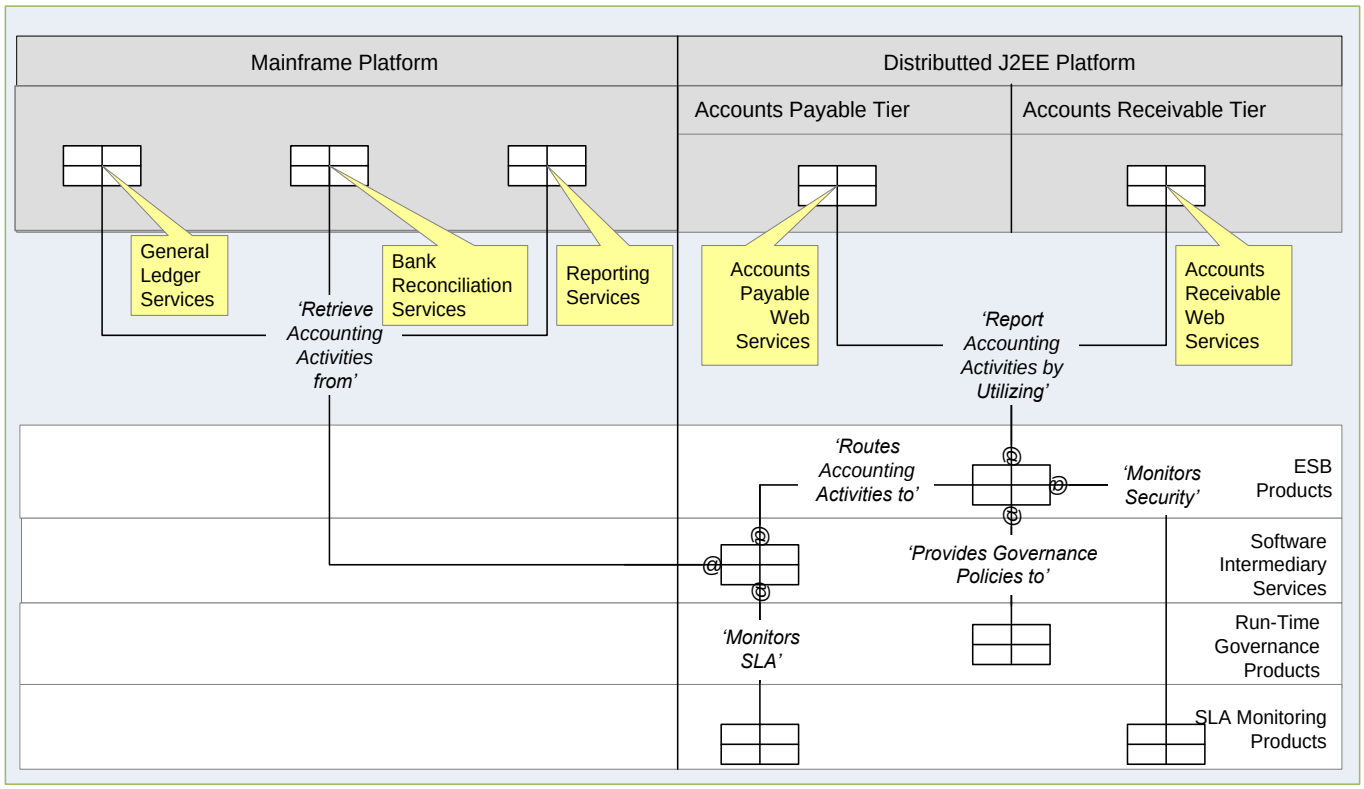

FIGURE 21: ASSET UTILIZATION DIAGRAM
WITH TWO PLATFORMS AND TWO TIERS

TRANSACTION DIRECTORY DIAGRAM (FIGURE 22)

a. Transaction Directory Section contains: Execute Trade, Issue Monthly Statement, Save Trading History, Open Trading Account
b. Asset Directory Section contains: Fixed-Income Trading Service Cluster, Equity Trading Portal, Mutual Funds Trading Application, Customer Support Web Services
c. Connector: Executed

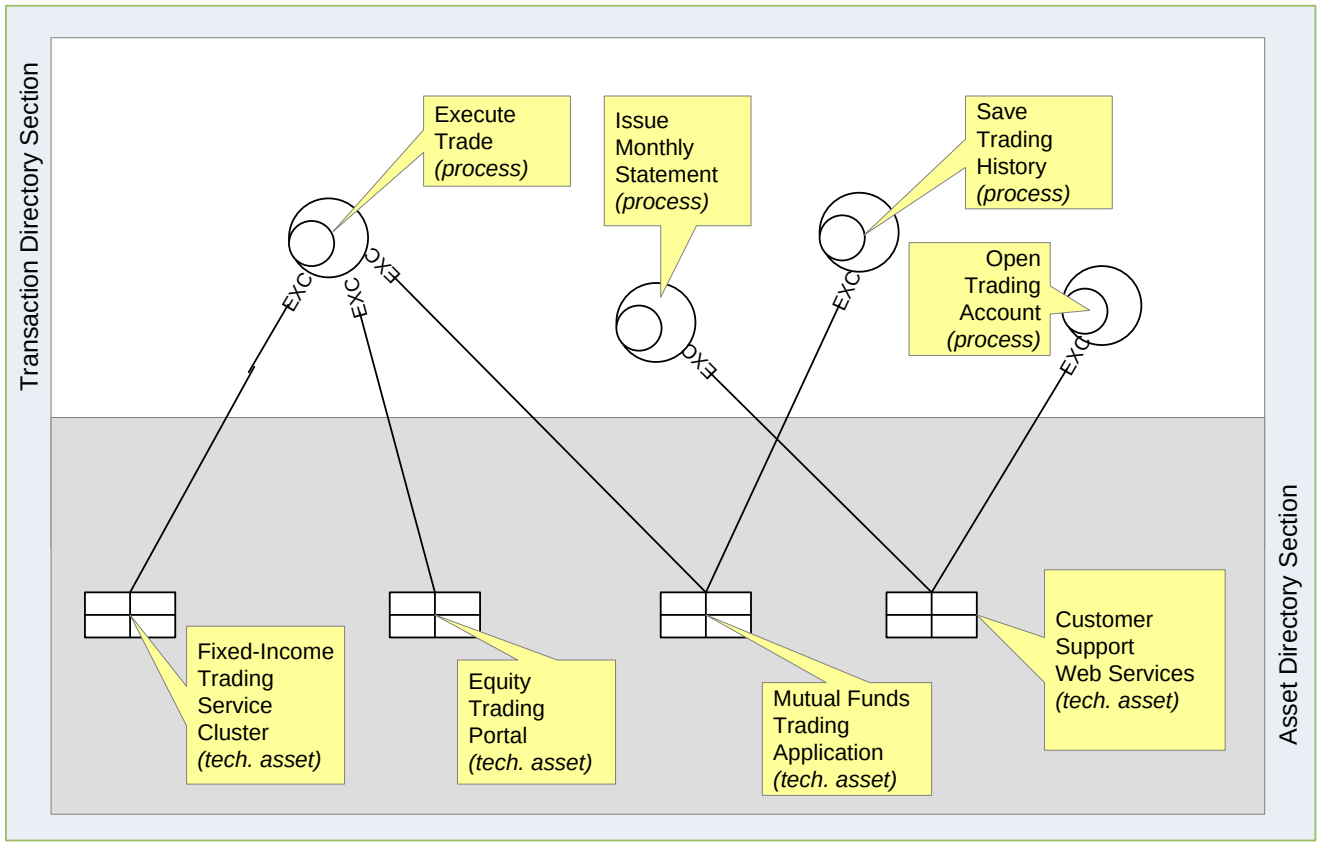

FIGURE 22: TRANSACTION DIRECTORY DIAGRAM
WITH TRANSACTION DIRECTORY SECTION
AND ASSET DIRECTORY SECTION