

T-DSCM: A Three-Dimensional Framework for Service Change Management in SOA Environments

DAVOOD BAHADORI & AHMAD ABDOLLAHZADEH BARFOROUSH
AMIRKABIR UNIVERSITY OF TECHNOLOGY

Outline

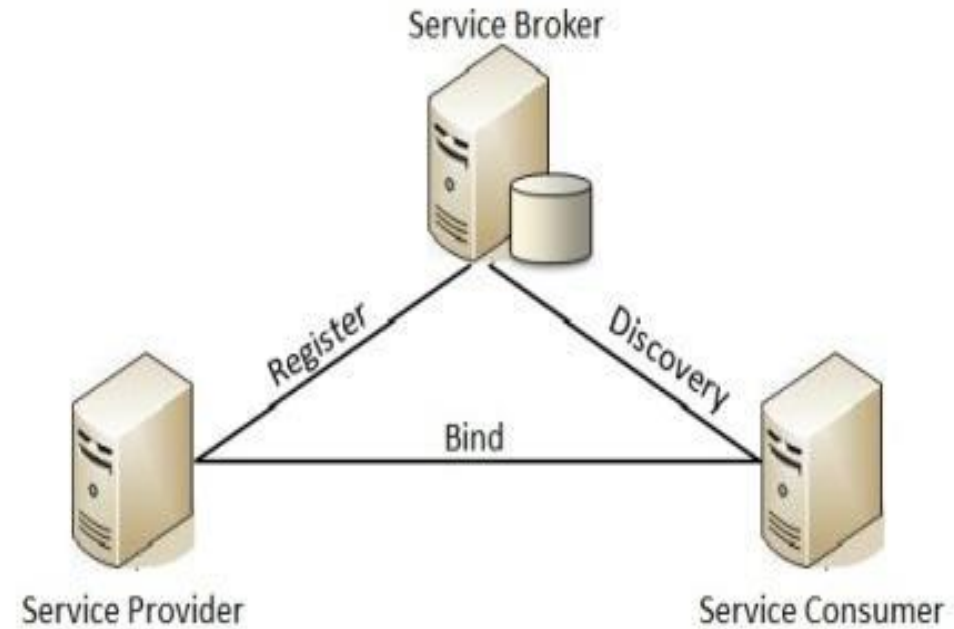
- ☐ Introduction
- ☐ Background
- ☐ Framework Overview
- ☐ Framework Evaluation

Introduction Service Oriented Architecture

- ❑ SOA is an architectural style, which encapsulates system or application function(s) into **service** as a **coarse-grained, discoverable** software entity and provides those **usable for public consumers**
- ❑ Today, in ICT and **Industry 4.0** domains, **SOA** and related technologies, as well as standards such as Enterprise Service Bus (ESB) are one of the key features of **reference architecture** due to providing **automated interoperability** and **real-time dynamic response to change** in the environment
- ❑ In SOA, services can be **developed** using a **variety of technologies**, which are different in design and implementation models
 - SOAP Web service & REST-full service
 - Object Management Group CORBA, RMI, and DCOM
 - Asynchronies services (e.g. IBM MQ)
 - ebXML

Introduction (Cont.) Service Oriented Architecture

- Main entities of SOA
 - Service Provider
 - Service Consumer
 - Service Broker



Introduction (Cont.)

Service Oriented Architecture

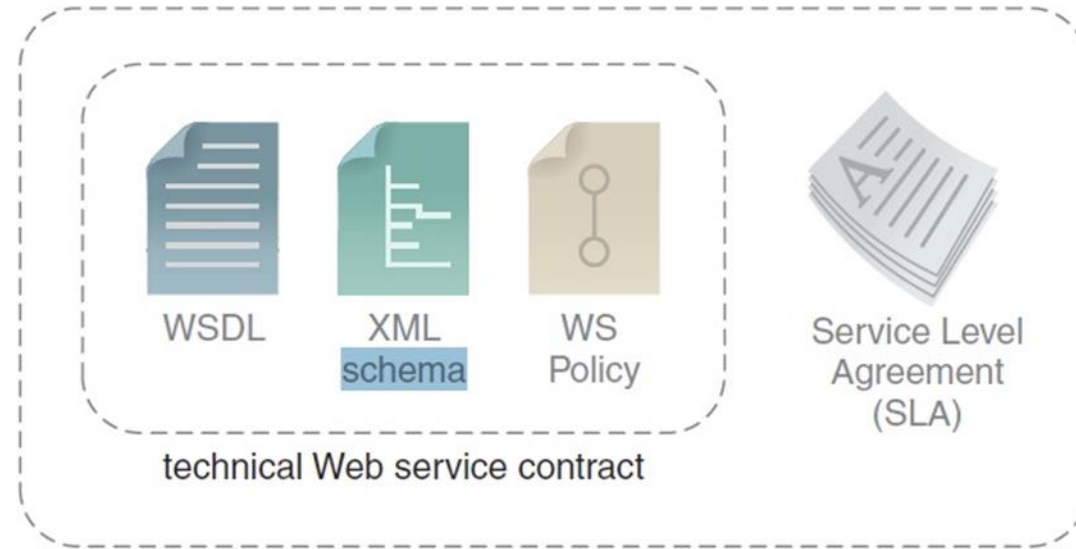
- ❑ Most widely technology to support the implementation of SOA is the **Web Service**
- ❑ Web services provide **more interoperable** and **loosely coupled** services as a result of using open standards and protocols such as XML, SOAP, UDDI, and WSDL
- ❑ SOAP is an **XML-based lightweight protocol** which used to exchange information in a decentralized and **distributed** environment
- ❑ The UDDI defines a standard method for both **publishing** and **discovering** web services
- ❑ WSDL is an **XML-based description language**, describing the **public interface** for web services

Introduction (Cont.)

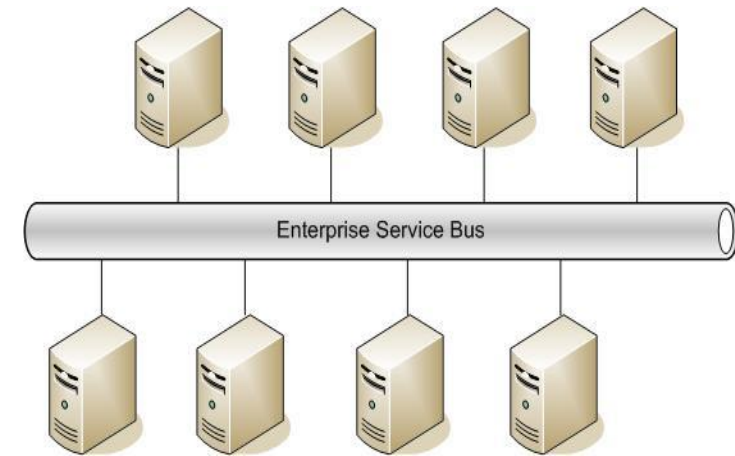
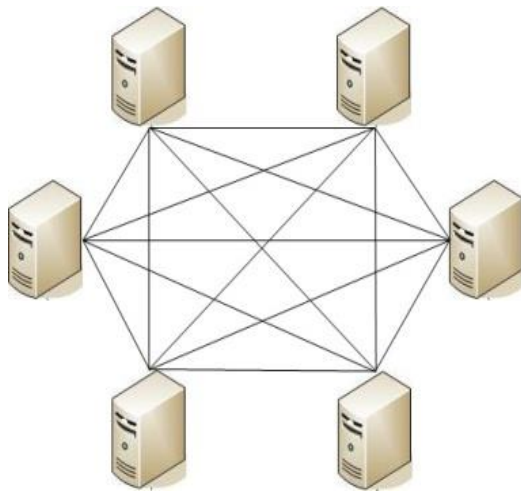
Web Service Contract

□ Parts of Web service contract

- WSDL
- XML Schema
- WS Policy
- SLA



Introduction (Cont.) Enterprise Service Bus

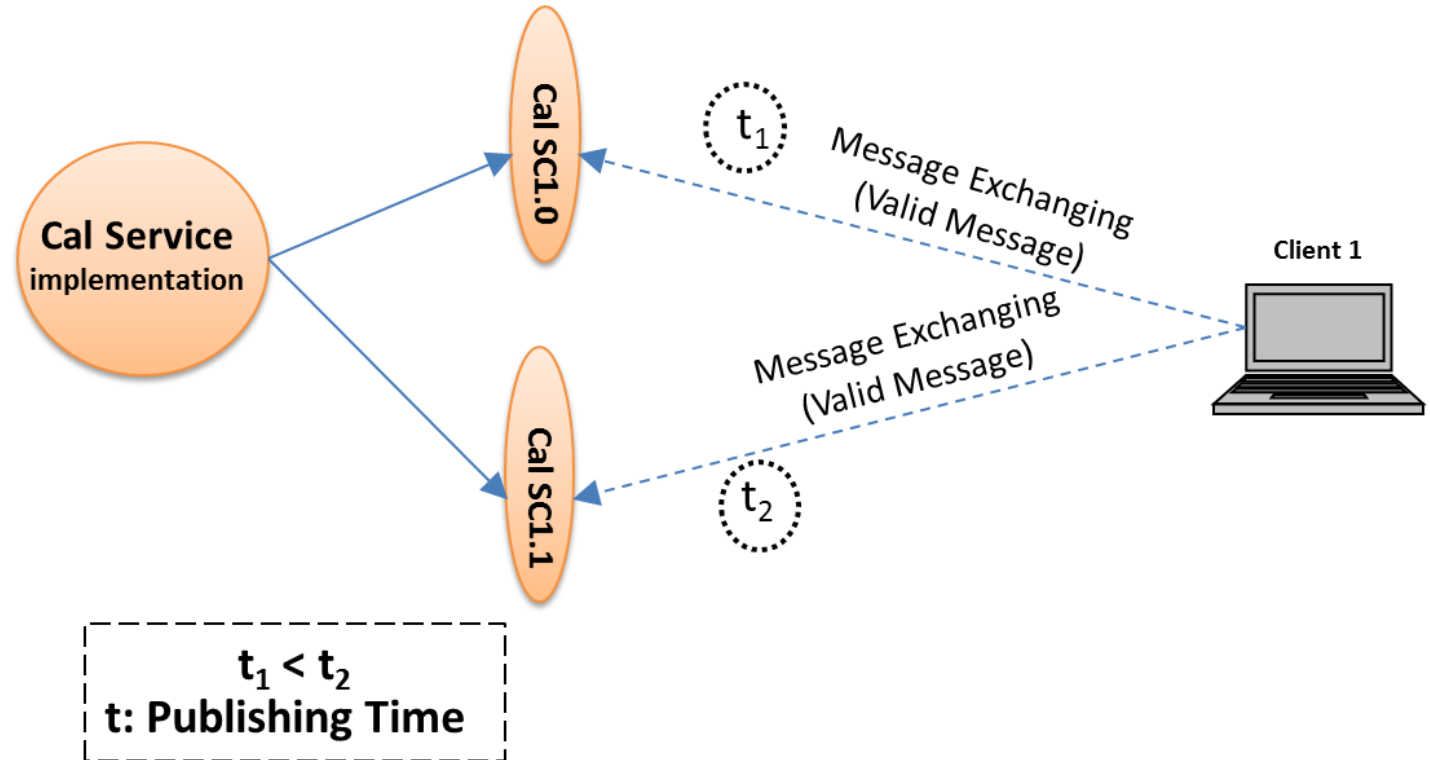


Background Compatibility and Versioning in Service

- it is a concept representing the type and severity of applied changes
- Compatibility can be considered as a metric for measuring the impact of service changes on other environmental entities
- Compatibility between different versions of services can be categorized into two general types
 - backward compatibility
 - forward compatibility

Background (Cont.) Backward compatibility

- ❑ Client1 bind to Cal Service and exchange message in t_1 using Cal SC 1.0 version
- ❑ Client1 still can exchange valid message with Cal Service after t_2 using Cal SC 1.1 version.
- ❑ There is no need any change to Client1 for exchanging valid message with Cal Service



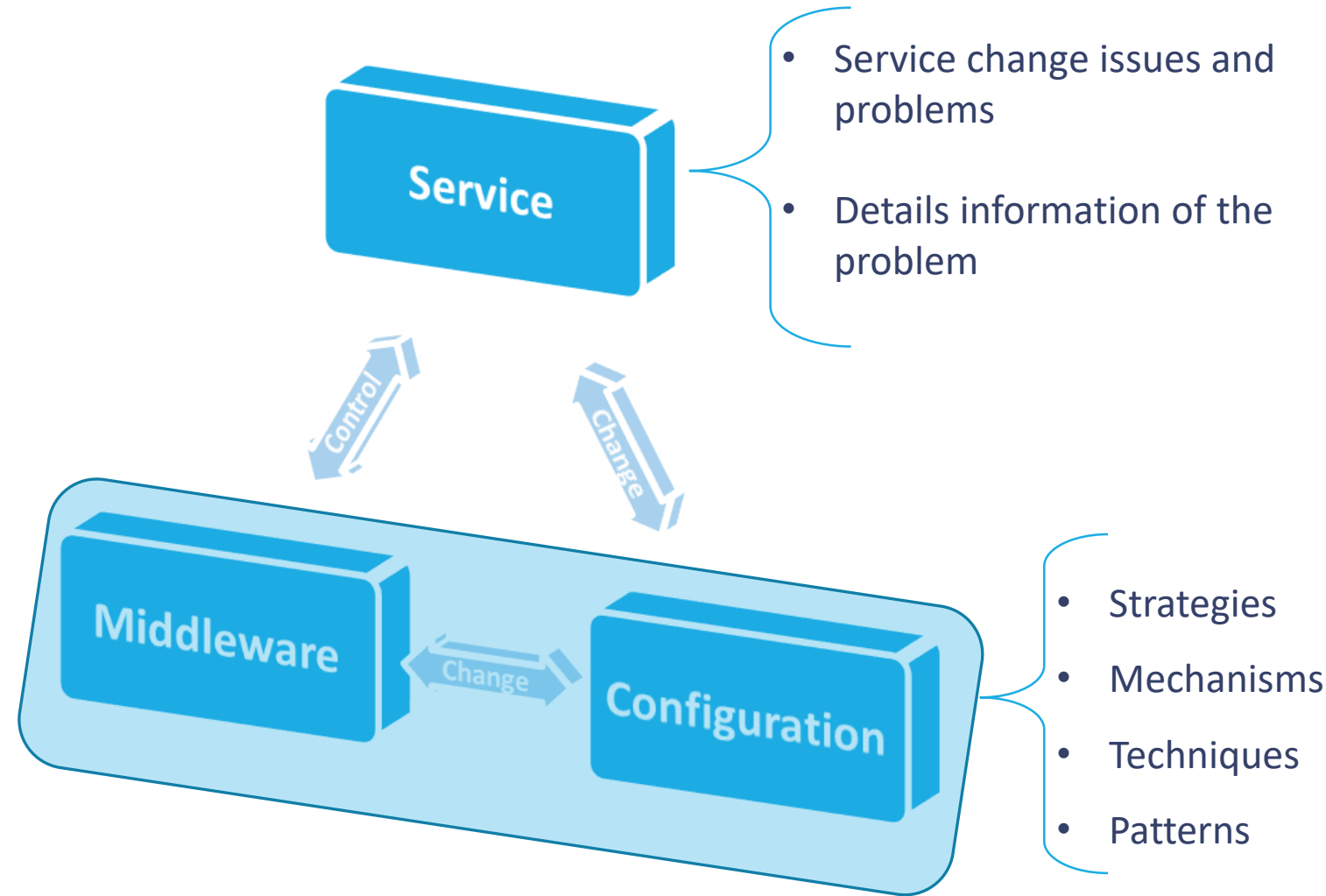
Background (Cont.)

Compatible and Incompatible Change, Valid Message

- ☐ The focus of this paper is on backward compatibility
- ☐ Changes that do not break the backward compatibility known as compatible changes
- ☐ If the applied change to a service has negative impact for the current user of the service, then this change is incompatible change
- ☐ A valid message is a message request that compatible with service technical interface, in otherwise is an invalid message

Framework Overview

- ❑ Three Dimensions:
 - ❑ Service
 - ❑ Configuration
 - ❑ Middleware
- ❑ All dimensions could have effects on each other and activate some system capabilities and functionalities
- ❑ Problem Domain: Service Dimension
- ❑ Solution Domain: Middleware and Configuration Dimensions



Framework Overview (Cont.) Service Dimension

- ☐ In Service Dimension the main problem should be defined
- ☐ Is the “***What dimension***” of the domain
- ☐ Focuses on the service change related problems and issues
 - Service evolution
 - Service versioning
 - Service availability and etc.
- ☐ Includes some activities in order to identify more details information about the problem
 - ☐ Type of the changes
 - ☐ Trigger of the changes
 - ☐ Potential destinations of the changes and etc.

Framework Overview (Cont.) Service Dimension Activities

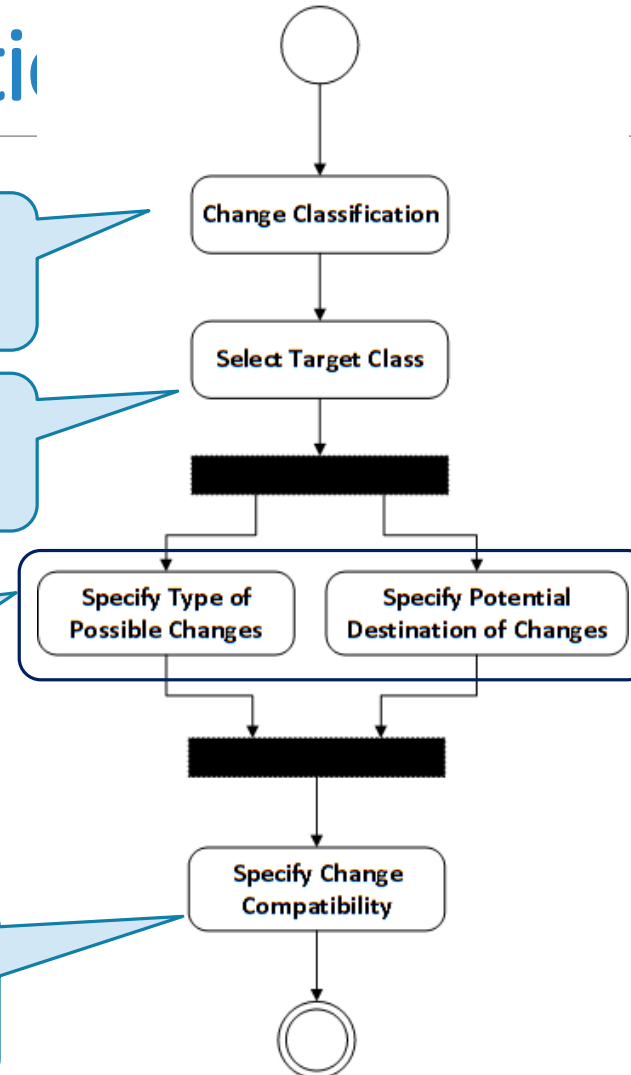
Classify all of the changes

Select candidate classes for managing

Specify details information of possible changes in each selected classes:

- Type of change
- Triggers of changes
- Potential destination of change

Specify the compatibility of changes for each possible combination of change type and change destination



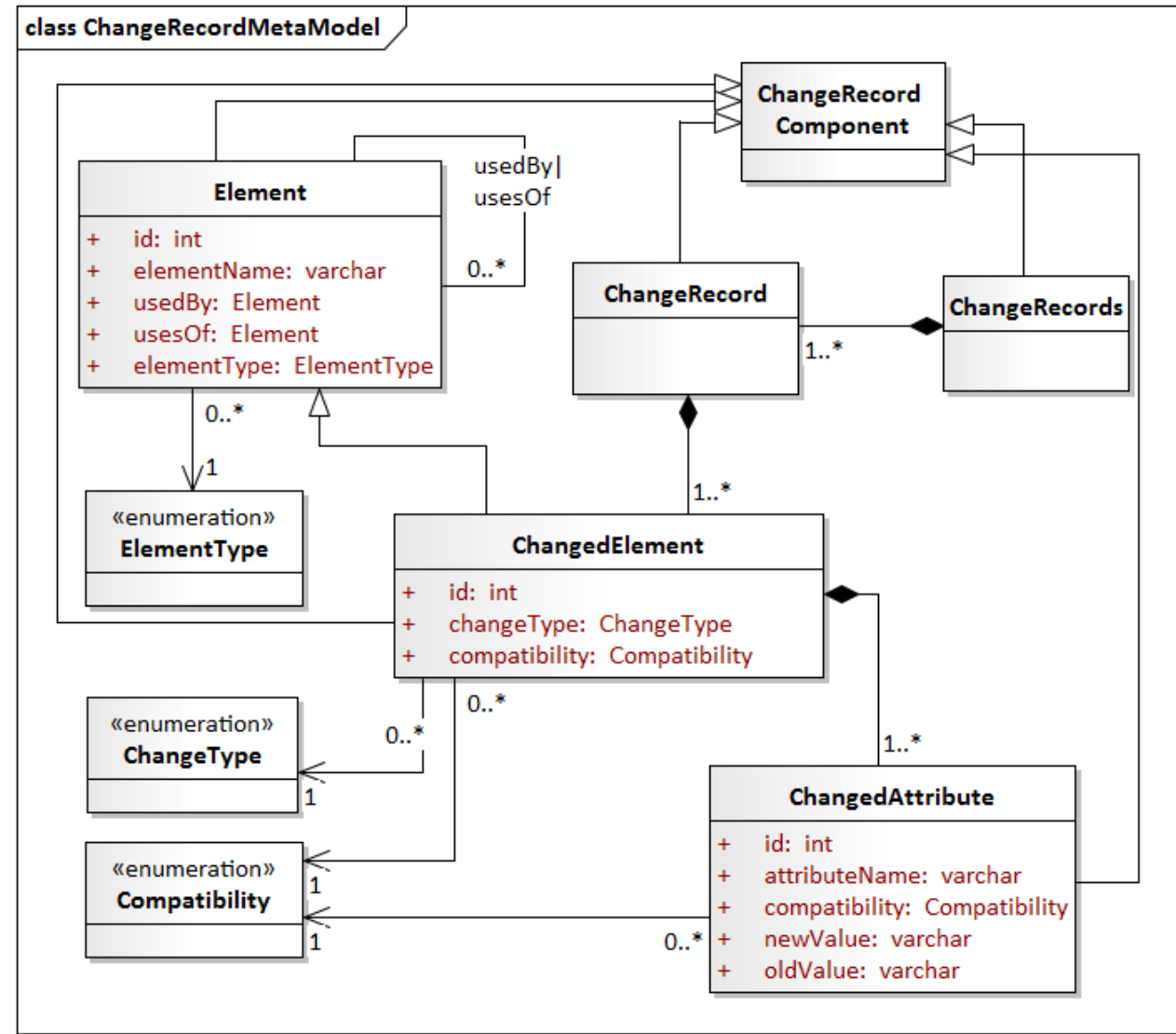
Framework Overview (Cont.)

Configuration Dimension

- ☐ Is the solution domain (dimension)
- ☐ The purpose of this dimension is to provide and prepare required information for Middleware Dimension to manage and control changes
- ☐ Access to versioning information can be realized using
 - ☐ Versioning tags in service contract (in service provider side)
 - ☐ Generating a compatibility message to consumer (in service provider side)
 - ☐ Calculating delta using comparing versions (in repository or consumer side)
- ☐ Need to well-form structure for storing the meta-data of the change record

Meta-model for Change Record

- ❑ Changed element: contains information about the destination of change as well as change type
- ❑ Dependency: to identify the dependency between elements
 - ❑ identify the propagation of changes
 - ❑ *UsedBy* attribute
 - ❑ *UsesOf* attribute
- ❑ Compatibility: in order to specify the compatibility type of changes



Framework Overview (Cont.) Configuration Dimension

Define change record incompatibility patterns by considering the compatibility information from service dimension.

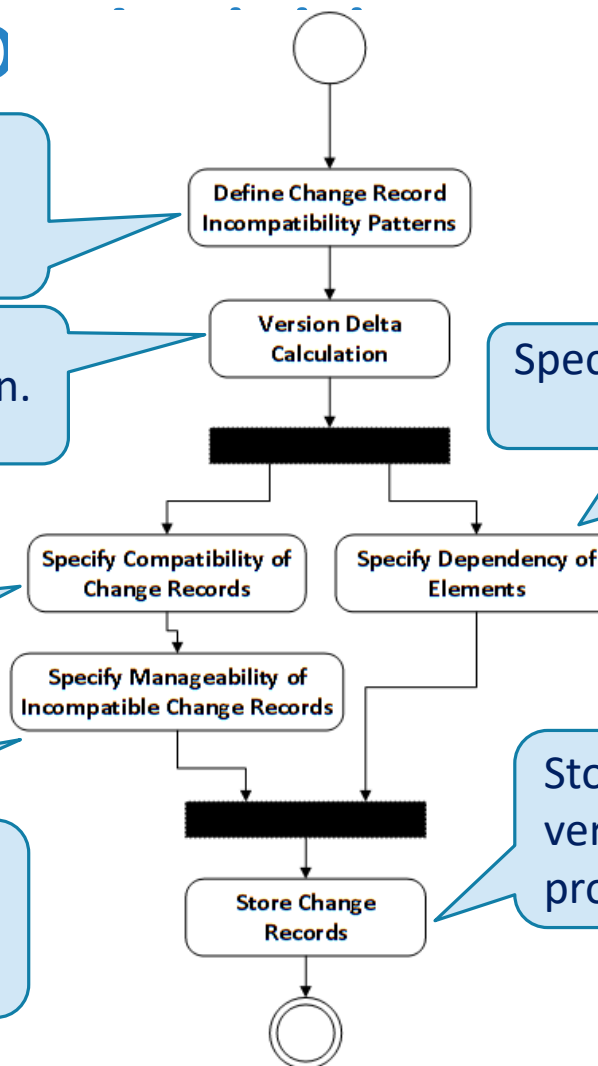
Calculate the change delta of the new service version.

Specify the compatibility type of occurred changes based on the defined incompatible patterns.

Specify the manageability of incompatible changes.

Specify dependency between elements.

Store the change record of new service version in the repository based on the proposed meta-model.



Framework Overview (Cont.) Middleware Dimension

- ☐ Is the solution domain (dimension)
- ☐ In this dimension, mediation patterns and mechanisms will be used to manage service change events
- ☐ For one or more incompatibility patterns, which specified in CD, we should
 - ☐ Create a change scenario
 - ☐ Define a group of one or more Enterprise Integration Pattern (EIP)
- ☐ Use of Enterprise Service Bus in order to implement the mediation mechanisms for preparing the incoming messages

Middleware Dimension (Cont.)

Strategies for Managing Incompatible Changes

□ There are two strategies for managing occurred change in services

1. **Message manipulation strategy:** the incompatible change can be managed with message manipulation using mediators mechanisms.
2. **Service replacement strategy:** the incompatible change cannot be managed with message manipulation and need to service replacement or exposing an error response.

Framework Overview: Middleware Dimension (Cont.)

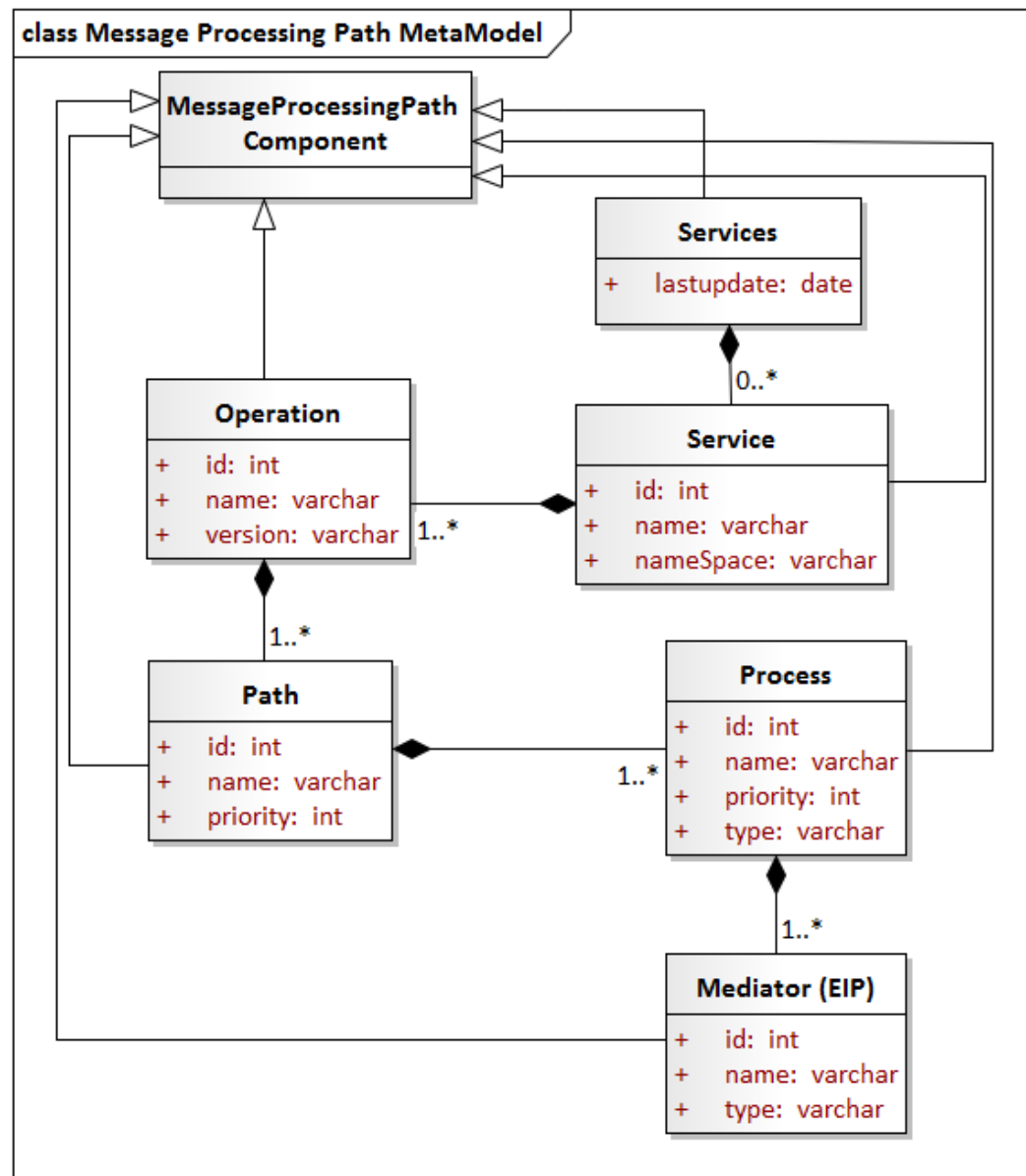
Message Processing Path Record

- ☐ Passing invalid request messages through processing steps in order to
 - ☐ Convert to valid message
 - ☐ Routing to proper alternative destination

- ☐ We need to meta-model for storing message processing path records

Message Processing Path MetaModel

- ❑ Services: contains all services that have changed up to now.
- ❑ Service: represents a service that is subjected to changes.
- ❑ Operation: represents the operation of service that are subjected to change.
- ❑ Path: contains one or more process components in order to generate a path for message processing
- ❑ Process: defines mediation processes



Framework Overview: Middleware Dimension Message Preparation General Pattern (MPGP)

- ❑ MPGP is a general pattern for preparing incoming messages in order to deliver a valid message to destination service (changed service or alternative service)
- ❑ Makes the main body of the message preparation process
- ❑ Drives from routing slip pattern
- ❑ Include 5 steps to prepare the incoming messages

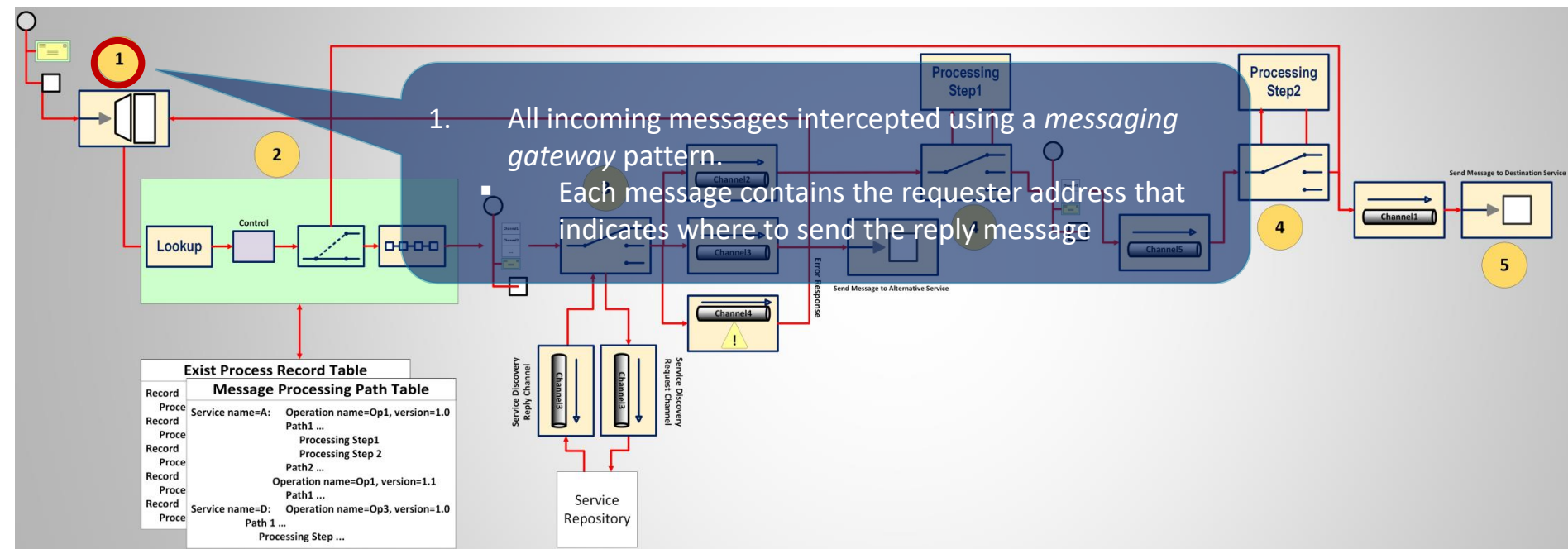
Framework Overview: Middleware Dimension

Message Preparation General Pattern (MPGP)

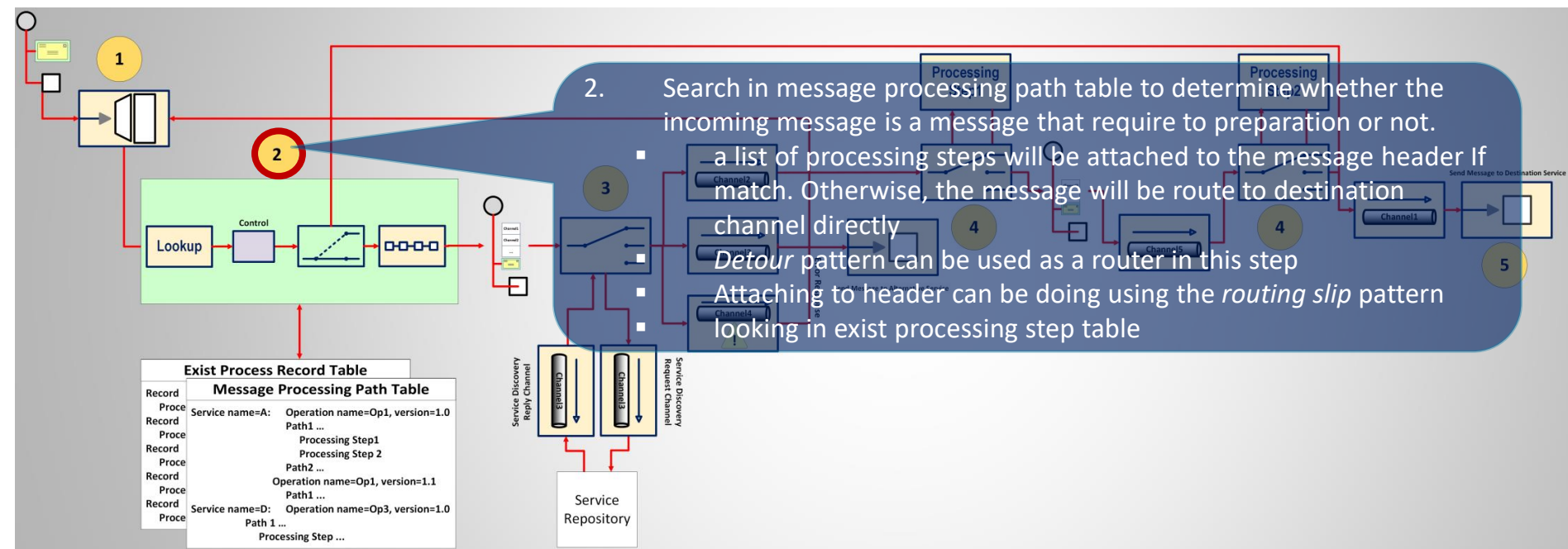
Message Preparation General Pattern (MPGP)

- ❑ **Problem:** the changed service may publish a new version that is incompatible with previous versions. So, the request messages from clients that bound to it before of change event may have an invalid form and resulting to service failure.
- ❑ **Intent:** convert a request message with targeted to changed service from invalid form to valid form using EIP patterns. So, the message will be prepared for delivery to changed service or alternative service in order to provide continuous service delivery.

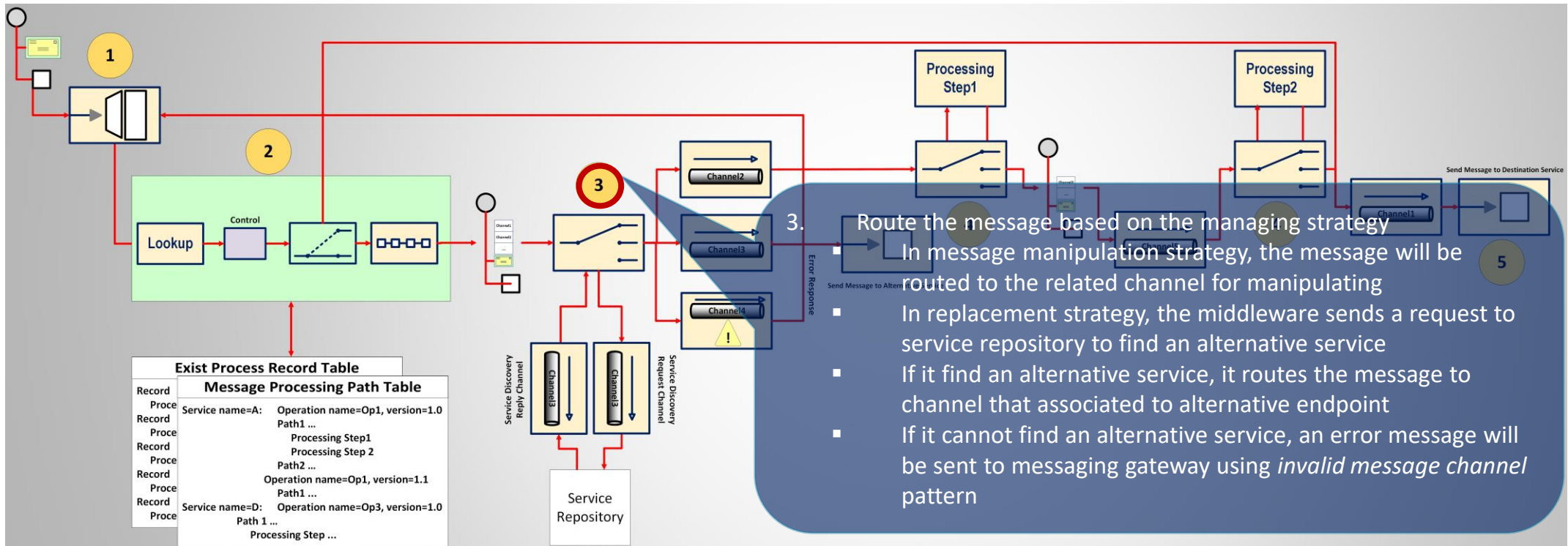
Framework Overview: Middleware Dimension Message Preparation General Pattern (MPGP)



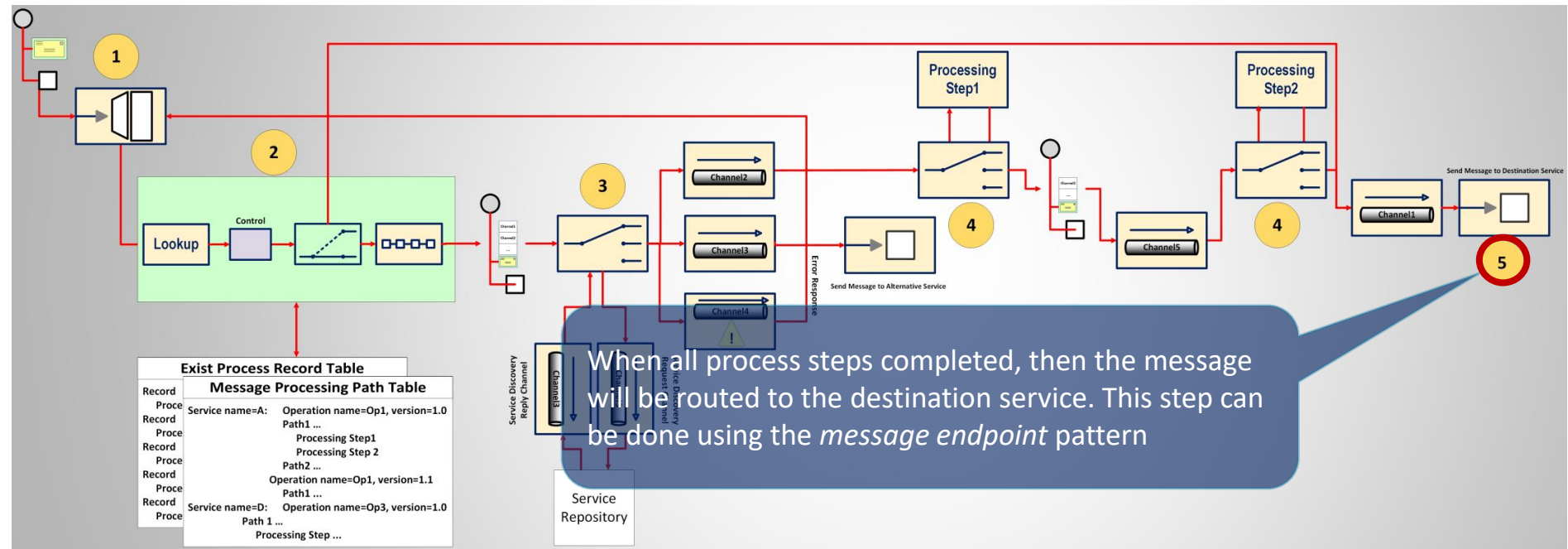
Framework Overview: Middleware Dimension Message Preparation General Pattern (MPGP)



Framework Overview: Middleware Dimension Message Preparation General Pattern (MPGP)



Framework Overview: Middleware Dimension Message Preparation General Pattern (MPGP)



Framework Overview: Middleware Dimension Activities

☐ Design time activities

1. Specify which incompatible records need to service replacement strategy.
2. Define change scenarios that must be applied to the incoming messages based on the incompatibility patterns.
3. Specify EIP pattern corresponding to each change scenario.
4. Specify ESB mechanisms in order to implement each EIP pattern in ESB.

Framework Overview: Middleware Dimension Activities

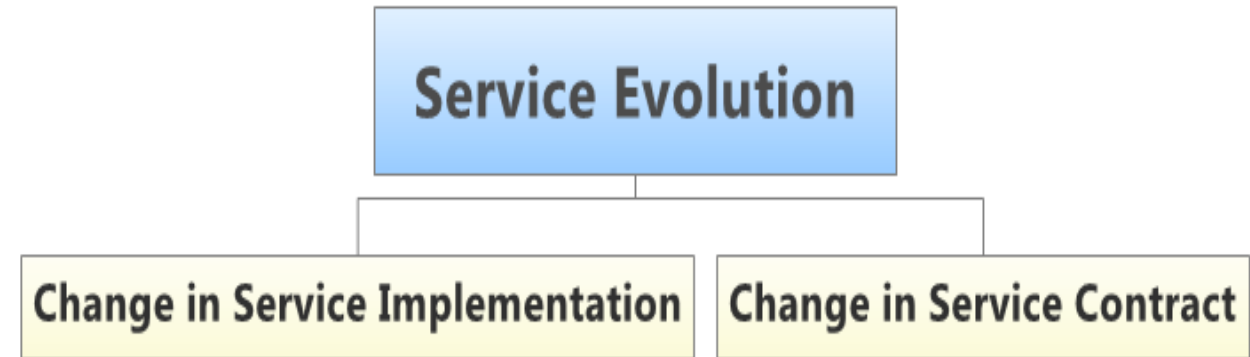
☐ Runtime activities

1. Acquiring the change records from configuration dimension
2. Define message processing path records and processing steps based on the acquired change records.
3. Intercept request messages and specify which of them should be sent to the changed services by looking up in message processing path table.

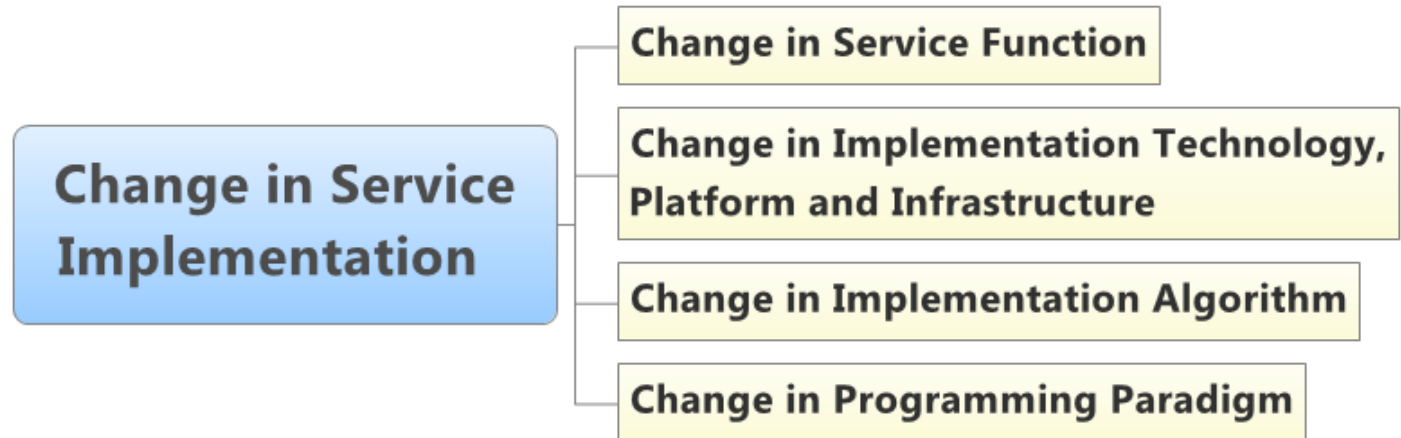
Framework Evaluation

- ❑ Implement a **version-aware SOA system** in order to prevent service failure when a new version of service is exposed
- ❑ After classifying the different type of changes in SOA based systems we focus on the changes which can cause to **service failure**
- ❑ One of the issues threatening service availability and service continuous delivery in such systems is **service contract change**
- ❑ the problem that we address using this framework is **continuous service delivery** in situations that **service contract change event occurs**
- ❑ implement two solution dimensions using an open source SOA platform, **WSO2**

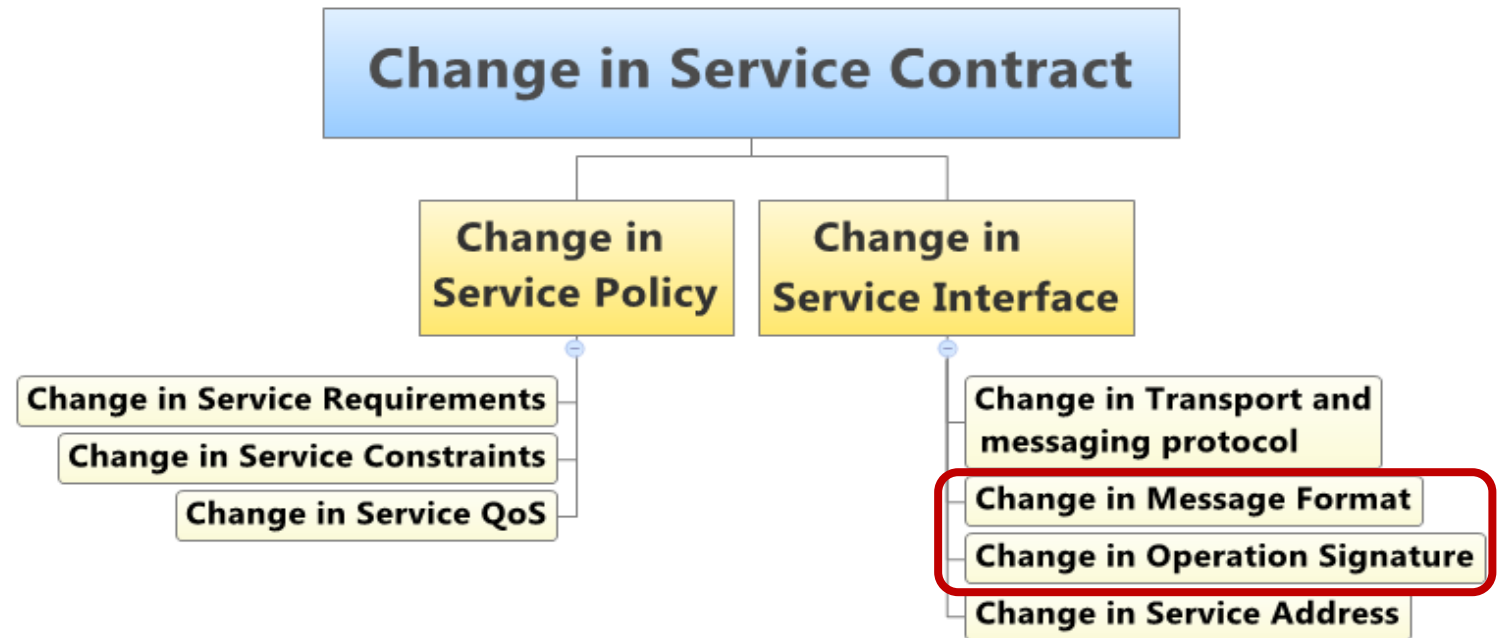
Framework Evaluation (Cont.) Service Dimension



Framework Evaluation (Cont.) Service Dimension

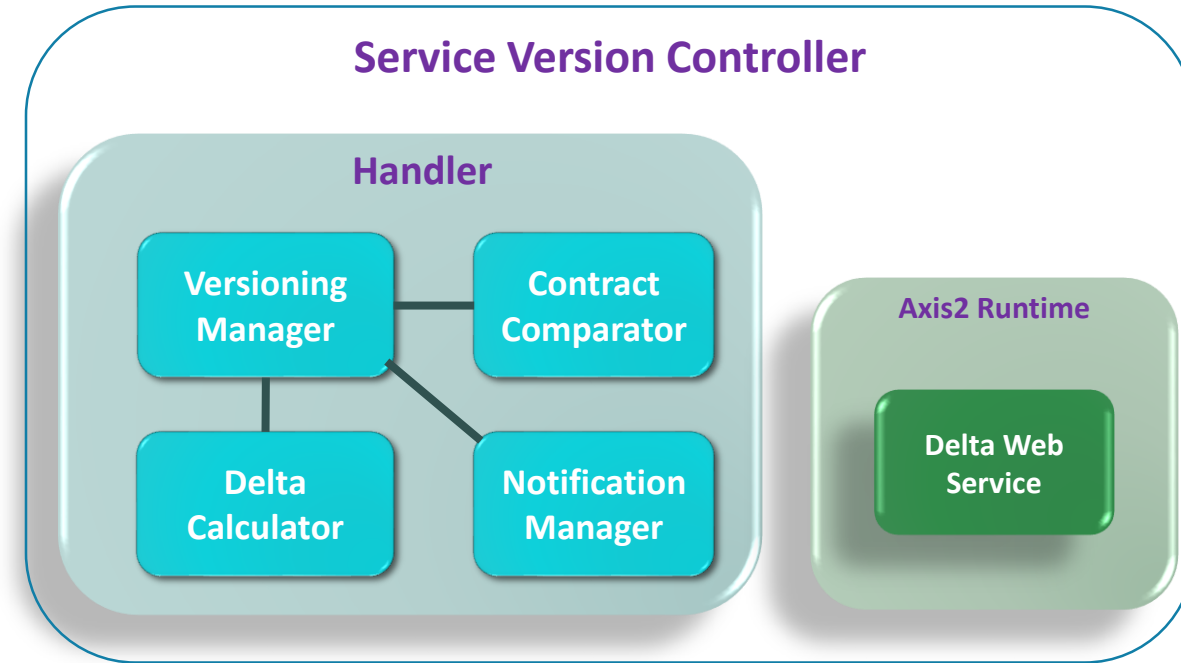


Framework Evaluation (Cont.) Service Dimension



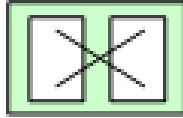

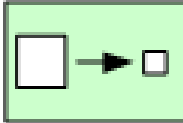

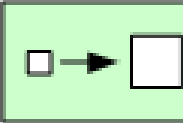

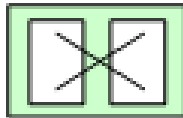

Framework Evaluation (Cont.) Configuration Dimension

- We create a handler for managing service versioning that is comprised of four component including
- **contract comparator**: for parsing and comparing each new version of the contract with previous versions,
 - **delta calculator**: for calculating the delta of versions and storing them in change record format
 - **notification manager**: in order to notify all entities which subscribe to inform about change, exposing new service version, and
 - **versioning manager**: acts as orchestrator that control and manage the interaction between other components in versioning handler

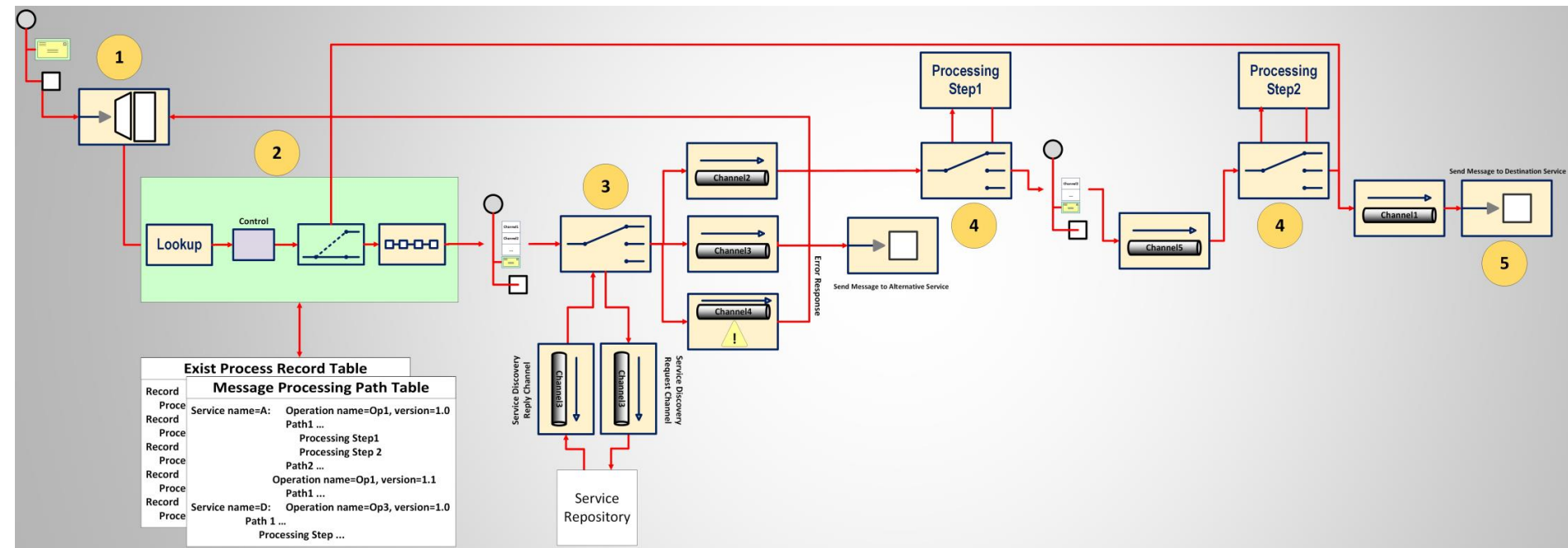


Framework Evaluation (Cont.) Middleware Dimension

- Use of WSO2 ESB to implement mediation mechanisms

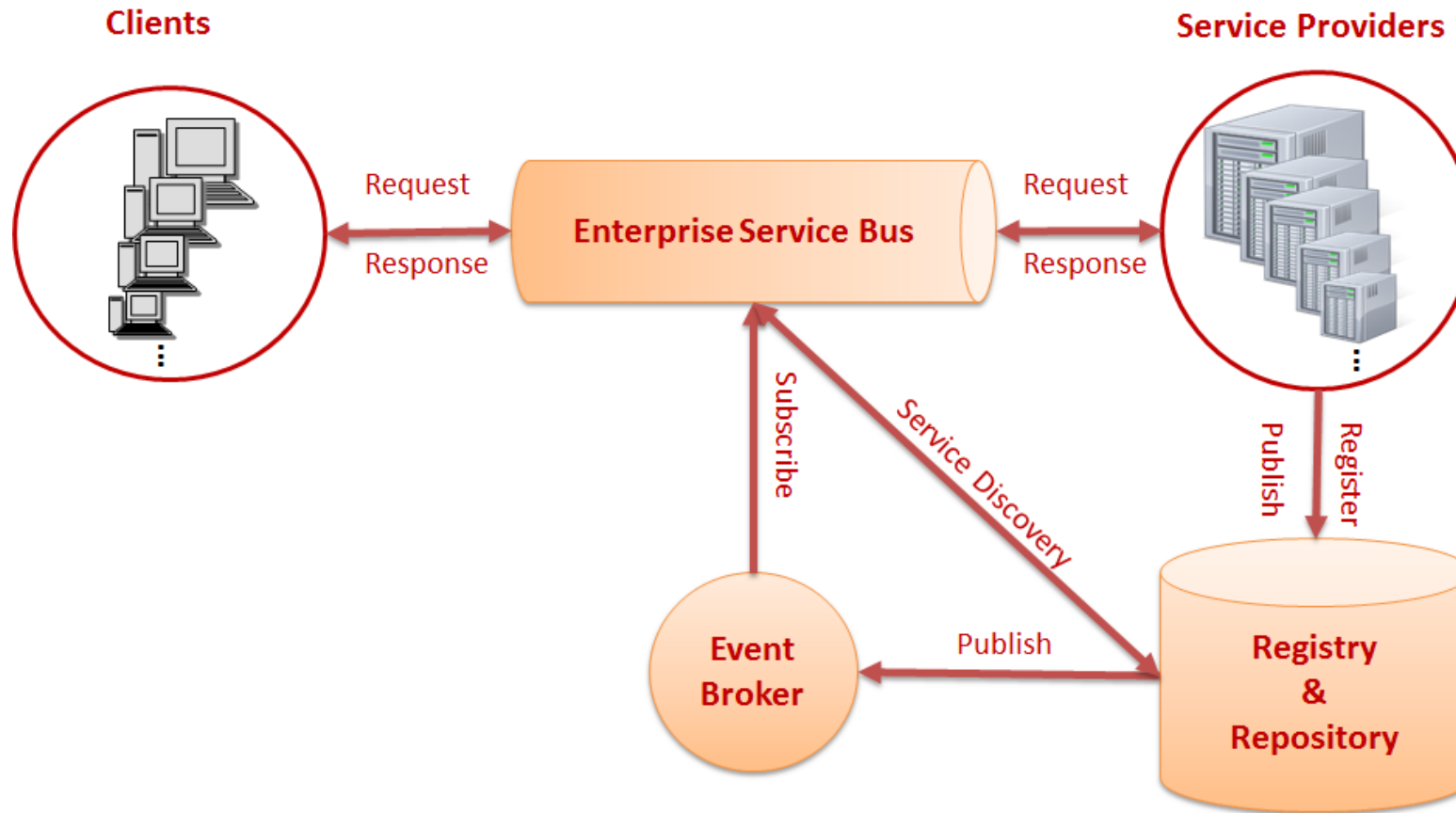
| Change Scenario | EIP Pattern | WSO2 Mediator |
|---|---|---|
| Changed the value of message element attributes |  Message Translator |  XSLT/PayloadFactory Mediator |
| Removed one or more element(s) of message |  Content Filter |  XSLT Mediator |
| Added one or more element(s) to message |  Content Enrich |  Enrich Mediator |
| Changed the order of message elements |  Message Translator |  XSLT/PayloadFactory Mediator |

Framework Evaluation (Cont.) Middleware Dimension

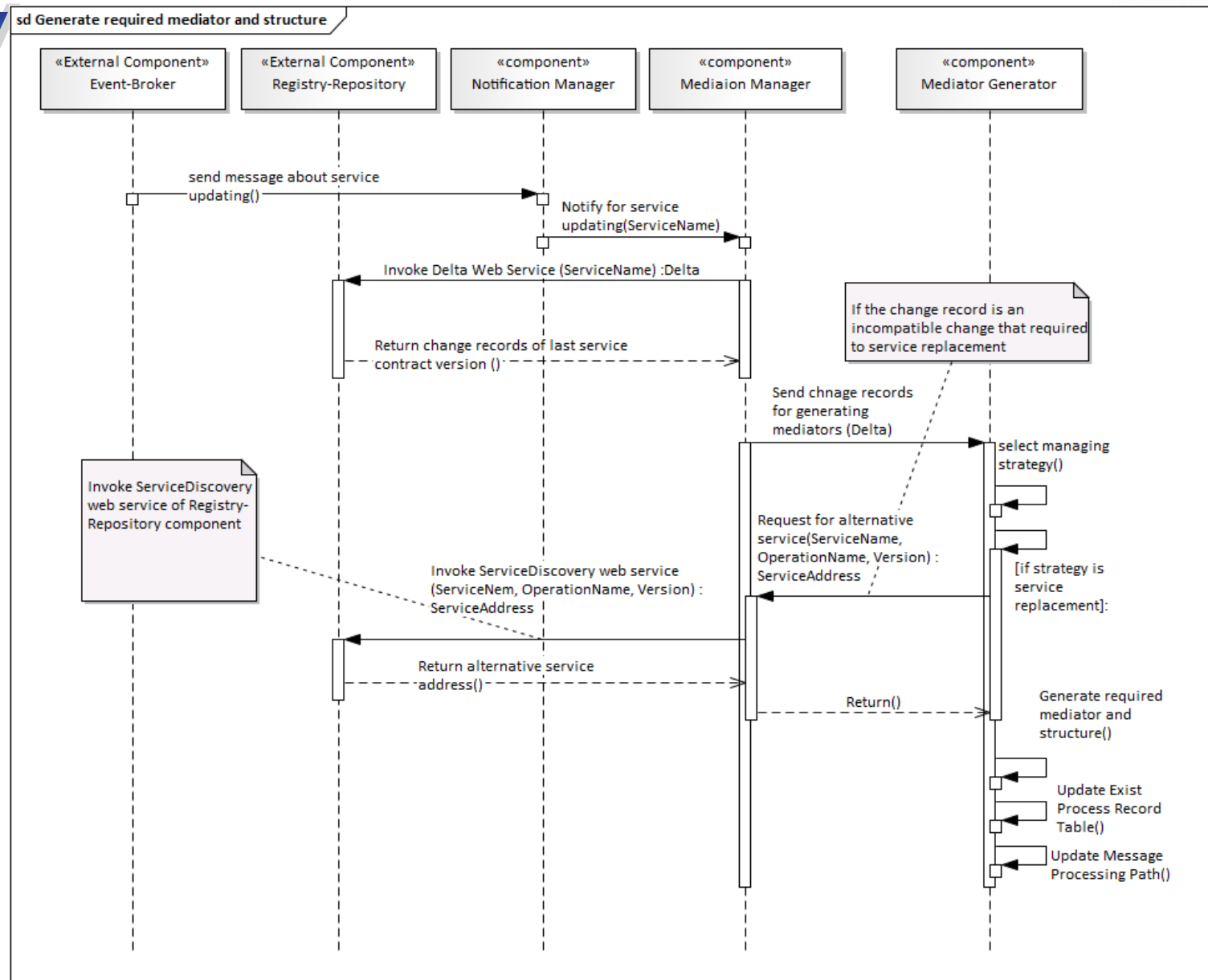


Framework Evaluation (Cont.)

Version-aware SOA System Architecture



Interaction Between Components



The End

