

Oracle Banking Digital Experience

Extensibility Guide

July 2017

ORACLE®

Extensibility Guide

July 2017

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax: +91 22 6718 3001

www.oracle.com/financialservices/

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Table of Contents

1.	Objective and Scope	4
2.	Overview of Use Cases	6
3.	Dictionary	18
4.	Outbound Webservice Extension	32
5.	Security Customizations	35

1. Objective and Scope

1.1 Background

CLIP is designed to help banks respond strategically to today's business challenges, while also transforming their business models and processes to reduce operating costs and improve productivity across both front and back offices. It is a one-stop solution for a bank that seeks to leverage Oracle Fusion experience across its core banking operations across its retail and corporate offerings.

CLIP provides a unified yet scalable IT solution for a bank to manage its data and end-to-end business operations with an enriched user experience. It comprises pre-integrated enterprise applications leveraging and relying on the underlying Oracle Technology Stack to help reduce in-house integration and testing efforts.

1.1.1 Objective and Scope

While most product development can be accomplished via highly flexible system parameters and business rules, further competitive differentiation can be achieved via IT configuration & extension support. Time consuming, custom coding to enable region specific, site specific or bank specific customizations can be minimized by offering extension points and customization support which can be implemented by the bank and / or by partners.

Extensibility objective

CLIP when extended & customized by the Bank and / or Partners results in reduced dependence on Oracle. As a result of this, the Bank does not have to align plans with Oracle's release plans for getting certain customizations or product upgrades. The bank has the flexibility to choose and do the customizations themselves or have them done by partners.

One of the key considerations towards enabling extensibility in CLIP has been to ensure that the developed software can respond to future growth. This has been achieved by disciplined software development leading to cleaner dependencies, well defined interfaces and abstractions with corresponding reduction in high cohesion & coupling. Hence, the extensions are kept separate from Core – Bank can take advantage of CLIP Core upgrades as most extensions done for a previous release can sit directly on top of the upgraded version. This reduces testing effort thereby reducing overall costs of planning & taking up an upgrade. This would also improve TTM significantly as the bank enjoys the advantage of getting universal features through upgrades.

The broad guiding principles w.r.t. providing extensibility in CLIP are summarized below:

- Strategic intent for enabling customers and partners to extend the application.
- Internal development uses the same principles for client specific customizations.
- Localization packs.
- Extensions by Oracle Consultants, Oracle Partners, Banks or Bank Partners.
- Extensions through the addition of new functionality or modification of existing functionality.
- Planned focus on this area of the application.
- Standards based.
- Leverage large development pool for standards based technology.
- Developer tool sets provided for as part of JDeveloper and Eclipse for productivity.

Document Scope

The scope of this document is to explain the **customization & extension** of CLIP for the following use cases:

- Customizing CLIP UI
- Adding a new field or a table on the screen
- Removing fields from the UI
- Customizing CLIP application services and implement composite application services
- Adding pre-processing or post processing validations in the application services extension
- Adding Business Logic in pre hook or post hook points in the application services extension
- Altering the product behaviour at customizations hooks provided as adapter calls in functional areas that are prone to change and in between modules that can be replaced (e.g. alerts, content management)
- Adding new fields to the CLIP domain model and including it on the corresponding screen.
- Adding a new report
- Adding a partner link or a human task to an existing process
- Adding new steps as a sub-process
- Adding or customizing facts and business rules in the application and configuring them for different modules
- Adding the processing of the uploaded files data
- Adding the feature of printing the receipt once the transaction is over
- Defining the security related access and authorization policies
- Defining different security related rules, validator and processing logics
- Customizing different functionalities like user search, role evaluation and limit exclusion in the application related to security

This document would be a useful tool for Oracle Consulting, bank IT and partners for customizing and extending the product.

1.1.2 Complementary documentation

The document is a developer's extensibility guide and does not intend to work as a replacement of the functional specification which would be the primary resource covering the following:

CLIP installation & configuration.

CLIP parameterization as part of implementation.

Functional solution and product user guide.

1.1.3 Out of scope

The scope of extensibility does not intend to suggest that CLIP is forward compatible.

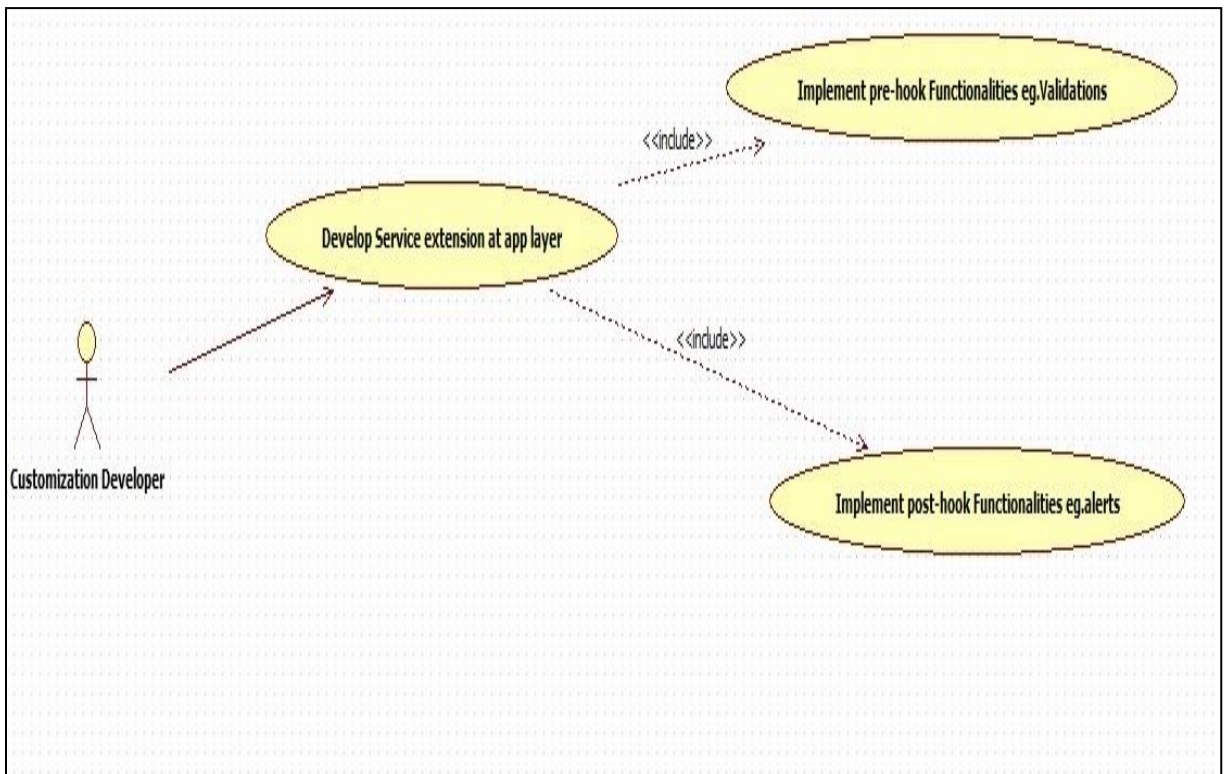
2. Overview of Use Cases

The use cases that are covered in this document shall enable the developer in applying the discipline of extensibility to CLIP. While the overall support for customizations is complete in most respects, the same is not a replacement for implementing a disciplined, thoughtful and well designed approach towards implementing extensions & customizations to the product.

2.1 Extensibility use cases

This section gives an overview of the extensibility topics and customization use cases to be covered in this document. Each of these topics will be detailed in the further sections.

- Extending Service Execution



- In CLIP, additional business logic might be required for certain services. This additional logic is not part of the digital experience product functionality but could be a client requirement. For these purposes, hooks have been provided in the application code wherein additional business logic can be added or overridden with custom business logic. The hook provided is:
 - Service Extensions

This hook resides in the **app** layer of the application service. This hook is present for both before as well after the actual service execution. The additional business logic has to implement the interface **I<service_name>ApplicationServiceExt** and extend and override the default implementation **Void<service_name>ApplicationServiceExt** provided for the service. Multiple implementations can be defined for a particular service. The service extensions

executor invokes all the implementations defined for the particular service both before and after the actual service executes.

2.2 Extending Service Executions

This section describes how additional business logic can be added prior to (pre hook) and / or post the execution (post hook) of particular application service business logic on the host side. Extension prior to a service execution can be required for the purposes of additional input validation, input manipulation, custom logging etc. A few examples in which the application service extensions in the form of pre and post hook could be required are mentioned below.

An application service extension in the form of a pre hook can be important in the following scenarios:

- Additional input validations
- Execution of business logic, which necessarily has to happen before going ahead with normal service execution.

An application service extension in the form of a post hook can be important in the following scenarios:

- Output response manipulation
- Custom data logging for subsequent processing or reporting.

The CLIP application provides one layer where the pre and post extension hooks for extending service execution can be implemented i.e.

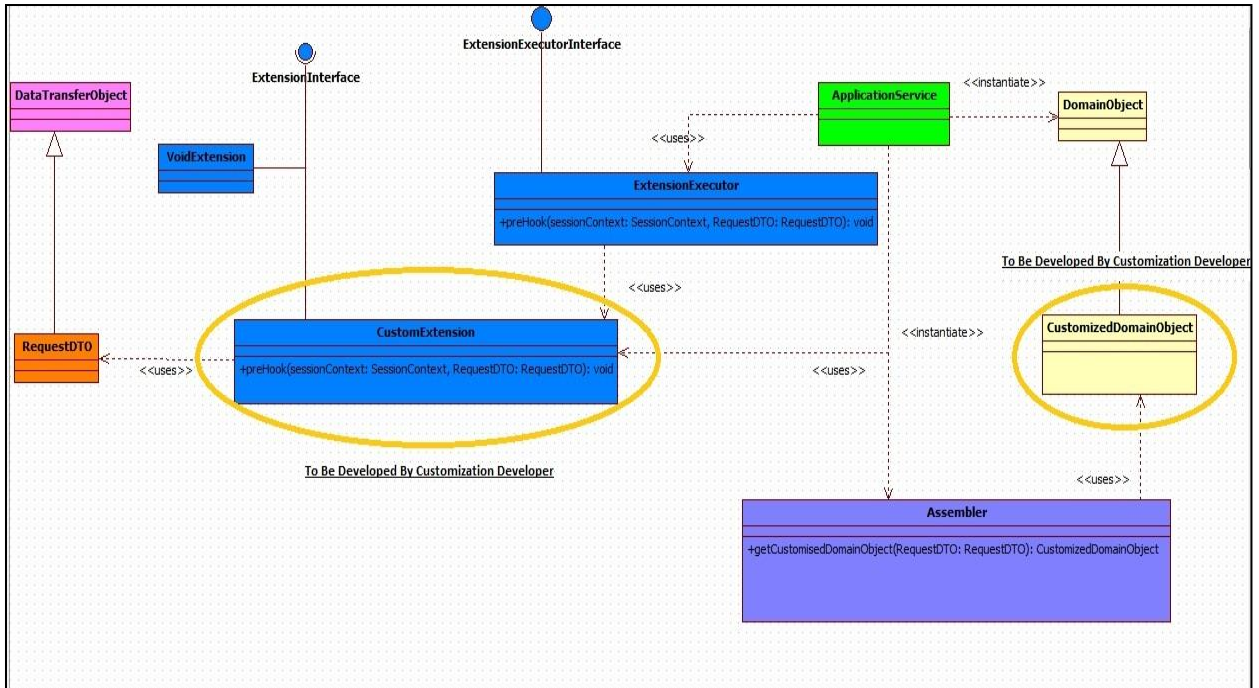
Application Service layer – referred to as the “app” layer extension.

2.3 Service Extension – Extending the “app” layer

The “app” layer is referred to as the application service layer and it denotes the business logic that executes as part of a service method exposed by CLIP middleware host. Extension points provided as pre & post hooks are present in this layer at the same points in the service.

2.3.1 Class Diagram

The Application Service layer involves the interaction between a set of components of the product mentioned in the class diagram below.



2.3.2 Data Transfer Object

Subclasses of the Customized Domain Object which are extended by the partners or consulting teams can be mapped as input & output to the application services with the help of this class. Additionally, this class is used to store the user defined field reference key generated and stored inside the tables mapped to the domain objects. This will help us to handle additional fields added at UI layer.


```

public abstract class DataTransferObject extends Validatable implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = -6584908885732656582L;
    /**
     * Subclasses of the Customized AbstractDomainObject corresponding to this
     * AbstractDomainObject<br>
     * are defined with the help of this attribute. This concept can be extended
     * to have joined or<br>
     * union subclass heirarchy in subsequent releases.
     */
    private Dictionary[] dictionaryArray;

    /**
     * Returns data for subclasses of the Customized Domain Object as name value
     * pair data with the<br>
     * name being a fact.
     *
     * @return
     */
    public Dictionary[] getDictionaryArray() {
        return dictionaryArray;
    }

    public void setDictionaryArray(Dictionary[] dictionaryArray) {
        this.dictionaryArray = dictionaryArray;
    }
}

```

2.3.3 Application Service Extension Interface:

This interface has a pair of pre and post method definitions for each application service method of the present. A service extension class has to implement this interface. The **pre** method of the extension is executed before the actual service method and the **post** method of the extension is executed after the service method. The signatures of these methods are:

```
public void pre<Method_Name>(<Method_Parameters>) throws Exception;
```

```
public void post<Method_Name>(<Method_Parameters>) throws Exception;
```

For example:

```

public interface ILoanApplicationExt {

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     */
    @throws Exception
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
        throws Exception;

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required after
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *     The instance of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
     *     request.
     *
     * @param loanApplicationResponseDTO
     *     The instance of type {@link LoanApplicationCreateResponseDTO} used for creating loan application
     *     response.
     *
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext,
        LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
        LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception;

    /**
     * Extension point for LoanApplication.update. The method is intended to keep all the extensions required before
     * updating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationUpdateRequestDTO
     *     The request DTO of type {@link LoanApplicationUpdateRequestDTO} used for updating loan application of
     *     any party.
     *
     * @throws Exception
     */
}

```

(i) Application Service Extension Executor Interface

The service extension executor class, on load, creates an instance each of all the extensions defined in the service extensions configuration file. If no extensions are defined for a particular service, the executor creates an instance of the default extension for the service. The executor also has a pair of **pre** and **post** methods for each method of the actual service. These methods in turn call the corresponding methods of all the extension classes defined for the service. The naming convention for the generated executor classes which enable “extension chaining” is as shown below:

Interface : `I<Application Service Qualifier>ApplicationServiceExtExecutor`

Implementation : `<Application Service Qualifier>ApplicationServiceExtExecutor`

For example:

```

public interface ILoanApplicationExtExecutor {

    /**
     * Executor point for LoanApplication.create. It executes the extensions available before creating loan applications
     * from the Service class {@code LoanApplicationApplicationService}.
     */
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
        throws Exception;

    /**
     * Executor point for LoanApplication.create. It executes the extensions available after creating loan applications
     * from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *     The request DTO of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
     *
     * @param loanApplicationCreateResponseDTO
     *     The response DTO of type {@link LoanApplicationCreateResponseDTO} returned by the service
     *
     * @throws Exception
     */
    public void postCreate(SessionContext sessionContext,
        LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
        LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception;

    /**
     * Executor point for LoanApplication.update. It executes the extensions available before updating loan applications
     * from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationUpdateRequestDTO
     *     The request DTO of type {@link LoanApplicationUpdateRequestDTO} used for updating loan application of
     *     any party.
     *
     * @throws Exception
     */
}

```

(ii) Void Extension (Default Extension)

Void<Service_Name>Ext. This class implements the aforementioned service extension interface without any business logic viz. the implemented methods are empty.

The default extension is a useful & convenient mechanism to implement the pre and / or post extension hooks for specific methods of an application service. Instead of implementing the entire interface, one should extend the default extension class and override only required methods with the additional business logic. Product developers DO NOT implement any logic, including product extension logic, inside the default extension classes. This is because the classes are auto-generated & reserved for product use and get overwritten as part of a bulk generation process.

For example:

```

import com.ofss.digx.app.origination.dto.submission.application.LoanApplicationCreateRequestDTO;

/**
 * Represents the Extension Interface for Loan applications Service. Extensions for initiating and updating loan
 * application, getting all the offer related information in the loan application, selecting and updating offers,
 * updating insurance and service charges related information, information about loan and re-payment instructions can be
 * specified in this class. Instance of the Interface of this class returns the extensions list required by the
 * application service Extension Executor Class. |
 */
public class VoidLoanApplicationExt implements ILoanApplicationExt {

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required before
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *     The request instance of type {@link LoanApplicationCreateRequestDTO} for creating loan application
     *     request.
     *
     * @throws Exception
     */
    @Override
    public void preCreate(SessionContext sessionContext, LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO)
        throws Exception {

    }

    /**
     * Extension point for LoanApplication.create. The method is intended to keep all the extensions required after
     * creating loan applications from the Service class {@code LoanApplicationApplicationService}.
     *
     * @param sessionContext
     *     The session context of request in the form of {@link SessionContext}.
     *
     * @param loanApplicationCreateRequestDTO
     *     The instance of type {@link LoanApplicationCreateRequestDTO} used for creating loan application
     *     request.
     *
     * @param loanApplicationResponseDTO
     *     The instance of type {@link LoanApplicationCreateResponseDTO} used for creating loan application
     *     response.
     *
     * @throws Exception
     */
    @Override
    public void postCreate(SessionContext sessionContext,
        LoanApplicationCreateRequestDTO loanApplicationCreateRequestDTO,
        LoanApplicationCreateResponseDTO loanApplicationResponseDTO) throws Exception {
    }
}

```

(iii) 3.1.1.5 Custom Extension (Example snippet to be developed by the customization Developer)

This Customized Service Extension Class implements methods of Application Service Extension Interface. This class contains pre hook and post hook point for the service. The **pre** method of this customized extension is executed before the actual service method and the **post** method of this is executed after the service method.

For Example:

```

private static final String THIS_COMPONENT_NAME = VoidCollaborationExtDemo.class.getName();

/**
 * Holds the instance of {@link MultiEntityLogger} used for sending messages on the console.
 */
private com.ofss.fc.infra.log.impl.MultiEntityLogger formatter = com.ofss.fc.infra.log.impl.MultiEntityLogger
    .getUniqueInstance();

/**
 * Instance of type {@link java.util.logging.Logger} used for Logging in Collaboration service.
 */
private static transient Logger logger = com.ofss.fc.infra.log.impl.MultiEntityLogger.getUniqueInstance()
    .getLogger(THIS_COMPONENT_NAME);

@Override
public void preCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // calling a custom class to check DTO integrity.
    this.checkCollaborationDTO();

    try{
        NameValuePairDTO[] valuePairDTO = new NameValuePairDTO[1];
        valuePairDTO[0] = new NameValuePairDTO("mobileCustomer", "9595959595", "String");
        valuePairDTO[0].setGenericName("com.ofss.digx.domain.collaboration.entity.customdemo.CustomCollaborationDomainObject.mobileCustomer");

        Dictionary[] dictionary = new Dictionary[1]; //array of dictionary
        dictionary[0] = new Dictionary();
        dictionary[0].setNameValuePairDTOArray(valuePairDTO);
        collaborationDTO.setDictionaryArray(dictionary);
    }catch(java.lang.Exception e)
    {
        logger.log(Level.FINE, formatter.formatMessage("Pre-Create extension implementation sample"));
    }
}

private void checkCollaborationDTO() {
    System.out.println("Sample validation performed");
}

@Override
public void postCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO,
    CollaborationResponseDTO collaborationResponseDTO) throws Exception {
    // TODO Auto-generated method stub
}

@Override
public void preRead(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // TODO Auto-generated method stub
}
}

```

(iv) Customized Domain Object (Example snippet to be developed by the customization Developer)

This newly created Customized Domain Object class extends Existing Domain Object Class. Mapping for same should be done in database as Customized Abstract Domain Object Configuration. This class contains additional fields added at UI layer and getter, setter for the same.

For Example:

```

package com.ofss.digx.domain.collaboration.entity.customdemo;

import com.ofss.digx.domain.collaboration.entity.Collaboration;

/**
 * custom domain objet to handle the changes ate UI or pre hoot level
 *
 */
public class CustomCollaborationDomainObject extends Collaboration{

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    /**
     * store the mobile number to call on.
     */
    private String mobileCustomer;

    /**
     * @return the mobileCustomer
     */
    public String getMobileCustomer() {
        return mobileCustomer;
    }

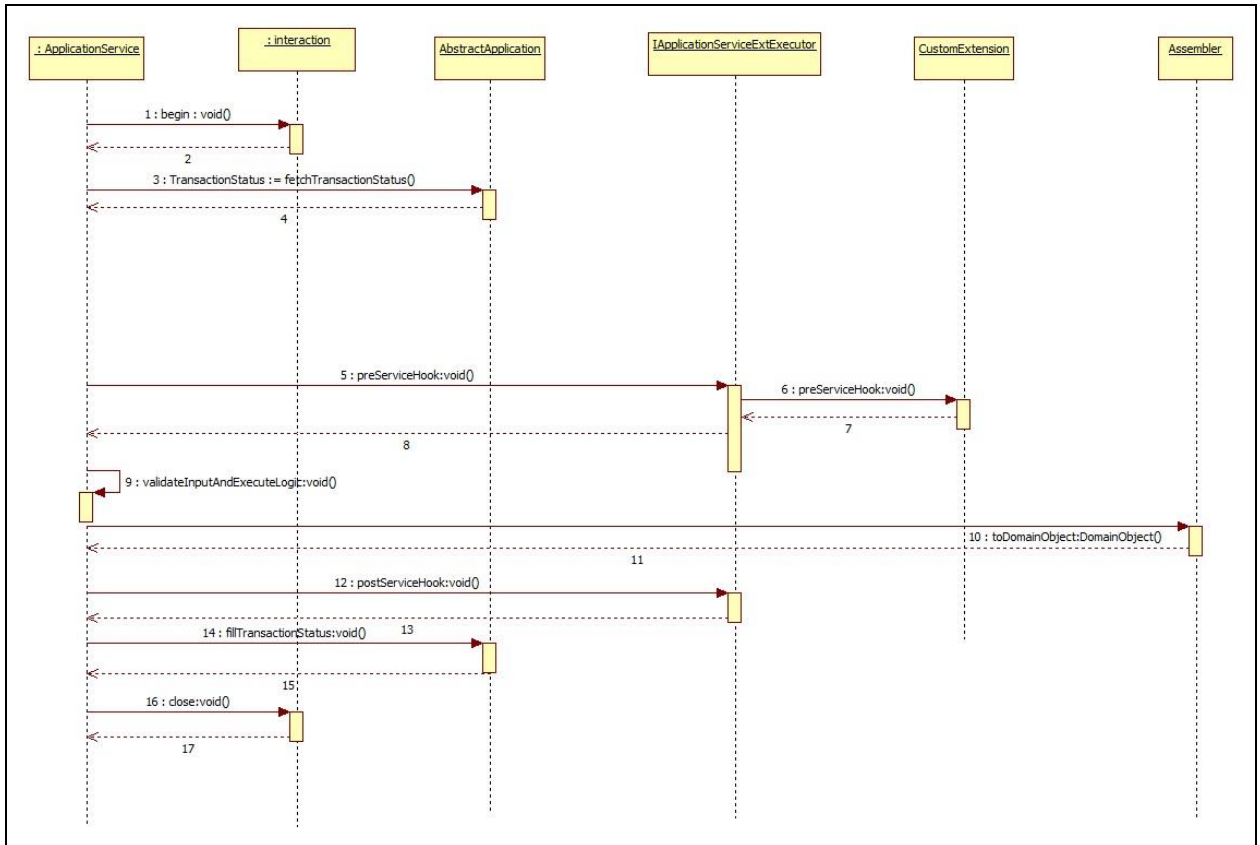
    /**
     * @param mobileCustomer the mobileCustomer to set
     */
    public void setmobileCustomer(String mobileCustomer) {
        this.mobileCustomer = mobileCustomer;
    }

}

```

2.3.4 Sequence Diagram

Every application service method has a standard set of framework method calls as shown in the sequence diagram below:



The pre hook is provided after the invocation of *fetchTransactionStatus* call inside the application service. At this step, the current task code is received, any additional manipulation of the input received from the User interface channel can be done in the pre hook. Apart from this additional data coming from the screen specific to client requirements can be handled in the pre hook.

The post hook is provided after the business logic corresponding to the application service invoked has executed and before the successful execution of the entire service is marked in the status object. This ensures that the status marking takes into consideration any execution failures of post hook prior to reporting the result to the calling source. Both, the pre and the post hooks accept the service input parameters as the inputs. The post hook also accepts the Response parameter as the input.

2.4 Extension Configuration

Set the property id and the property values in the **digx_fw_config_all_b** table. The property id will be the fully qualified name of the service and the value will be the fully qualified name of the custom extension created.

For example:

```

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG,
OBJECT_VERSION_NUMBER)
    
```

```
values ('com.ofss.digx.app.origination.service.submission.applicant.Applicant',
'ServiceExtensionsConfig',
'com.ofss.digx.app.origination.service.submission.application.ext.CustomLoanApplicationExtensi
on', 'N', 'asdf', 'asdf', 'asdf', ", 'asdf', ", 'Y', 1);
```

2.4.1 Mapping the domain object.

The domain object created needs to be mapped as a custom domain object for the existing domain object. For example:

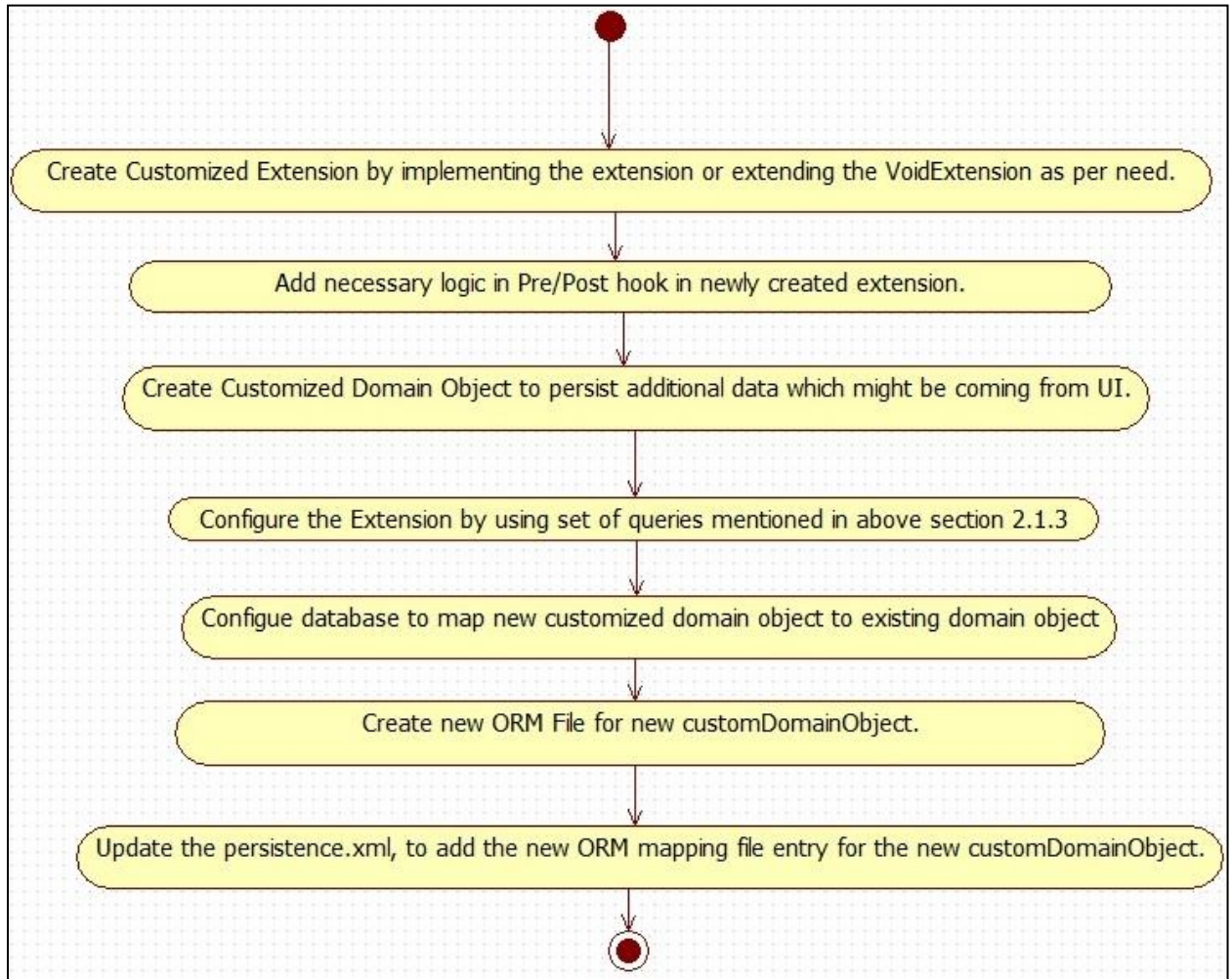
```
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG,
OBJECT_VERSION_NUMBER)
```

```
values ('com.ofss.digx.domain.origination.entity.submission.lending.application',
'CustomizedAbstractDomainObjectConfig',
'com.ofss.digx.domain.origination.entity.submission.lending.application.ext.Application', 'N', 'asdf',
'asdf', 'asdf', ", 'asdf', ", 'Y', 1);
```

Three main columns that need to be feed with new information are.

- CATEGORY_ID : “CustomizedAbstractDomainObjectConfig”
- PROP_VALUE:” CLASS NAME of the class implementing the custom domain object ”
- PROP_ID:” CLASS NAME OF THE DomainObject”.

2.5 Summary of Steps



3. Dictionary

Data transfer object (DTO) is a design pattern used to transfer data between an external system and the application service. All the information may be wrapped in a single DTO containing all the details and passed as input request as well as returned as an output response. The client can then invoke accessory (or getter) methods on the DTO to get the individual attribute values from the Transfer Object. All request response classes in CLIP application services are modelled as data transfer objects. These objects extend a base class *DataTransferObject* which holds an array of **Dictionary object**. The Dictionary encapsulates an array of **NameValuePairDTO** which is used to pass data of custom data fields or attributes from the UI layer to the host middleware.

Dictionary class looks like

```

010 Dictionary.class  CustomLoanApplicationExtension.java
1  package com.ofss.fc.framework.domain.common.dto;
2
3  import java.io.Serializable;
4
5  import javax.xml.bind.annotation.XmlType;
6
7  @XmlType(namespace="http://dto.common.domain.framework.fc.ofss.com")
8  public class Dictionary implements Serializable {
9
10     /**
11      *
12      */
13     private static final long serialVersionUID = 640766746819694755L;
14     private NameValuePairDTO[] nameValuePairDTOArray;
15
16     public NameValuePairDTO[] getNameValuePairDTOArray() {
17         return nameValuePairDTOArray;
18     }
19
20     public void setNameValuePairDTOArray(NameValuePairDTO[] nameValuePairDTOArray) {
21         this.nameValuePairDTOArray = nameValuePairDTOArray;
22     }
23
24
25 }
26

```

Following image shows use of dictionary with NameValuePairDTO and added it to the Data Transfer Object.

```

@Override
public void preCreate(SessionContext sessionContext, CollaborationDTO collaborationDTO) throws Exception {
    // calling a custom class to check DTO integrity.
    this.checkCollaborationDTO();

    try{
        NameValuePairDTO[] valuePairDTO = new NameValuePairDTO[1];
        valuePairDTO[0] = new NameValuePairDTO("mobileCustomer", "9595959595", "String");
        valuePairDTO[0].setGenericName("com.ofss.digx.domain.collaboration.entity.customdemo.CustomCollaborationDomainObject.mobileCustomer");

        Dictionary[] dictionary = new Dictionary[1]; //array of dictionary
        dictionary[0] = new Dictionary();
        dictionary[0].setNameValuePairDTOArray(valuePairDTO);
        collaborationDTO.setDictionaryArray(dictionary);
    }catch (java.lang.Exception e)
    {
        Logger.log(Level.FINE, formatter.formatMessage("Pre-Create extension implementation sample"));
    }
}
}

```

3.1 Translating Dictionary data into custom domain object

If dictionary is added to DTO then it is necessary to get customized domain Object which extends base Domain Object. Method **getCustomizedDomainObject** in **AbstractAssembler** is used for same.

Following image shows call to get Customized domain Object if additional data (Dictionary) is added to the request dto.

```

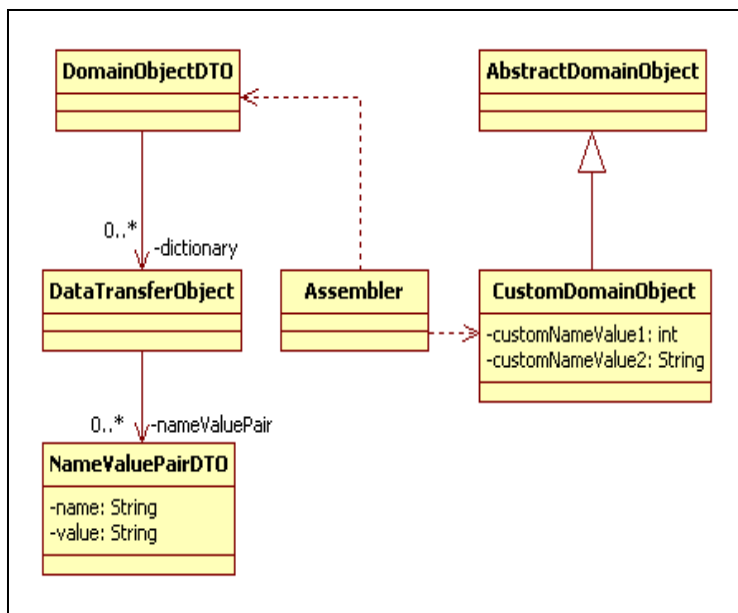
public Collaboration toDomainObject(CollaborationDTO collaborationDTO) {
    Collaboration collaboration;
    if (collaborationDTO.getDictionaryArray() != null) {
        try {
            collaboration = (Collaboration) getCustomizedDomainObject(collaborationDTO);
        } catch (java.lang.Exception e) {
            collaboration = new Collaboration();
        }
    } else {
        collaboration = new Collaboration();
    }

    CollaborationKey collaborationKey = new CollaborationKey();
    if (collaborationDTO.getId() != null) {
        collaborationKey.setId(collaborationDTO.getId());
    }
    collaboration.setExternalRefId(collaborationDTO.getExternalRefId());
    collaboration.setAuthorId(collaborationDTO.getAuthorId());
    collaboration.setAuthorName(collaborationDTO.getAuthorName());
    collaboration.setUrl(collaborationDTO.getUrl());
    collaboration.setExpiryDate(collaborationDTO.getExpiryDate());
    collaboration.setNoOfParticipants(collaborationDTO.getNoOfParticipants());
    collaboration.setCollaborationChannel(collaborationDTO.getCollaborationChannel());
    List<Participant> participantsList = new ArrayList<Participant>();
    if (collaborationDTO.getParticipants() != null && !collaborationDTO.getParticipants().isEmpty()) {
        for (ParticipantDTO participantDTO : collaborationDTO.getParticipants()) {
            Participant participant = new Participant();
            ParticipantKey participantKey = new ParticipantKey();
            participant.setKey(participantKey);
            participant.setInternalPartyId(participantDTO.getInternalPartyId());
            participant.setName(participantDTO.getName());
            participant.setPublishingId(participantDTO.getPublishingId());
            participant.setIsPublished(participantDTO.getIsPublished());
            participantsList.add(participant);
        }
    }
    collaboration.setParticipants(participantsList);
    collaboration.setCollaborationKey(collaborationKey);
    return collaboration;
}

```

3.2 Role of Dictionary in Extensibility

Following image shows how dictionary class is used and components with which it has Associativity.

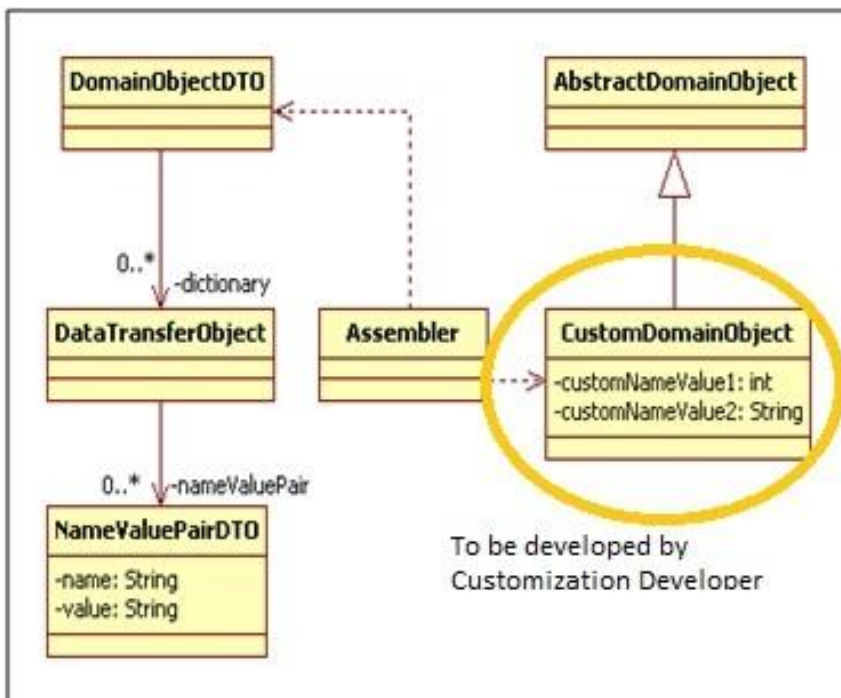


3.2.1 Domain Object Extension

This section describes how consultants or other third parties can extend domain and achieve Extensibility. This provides true domain model extension capabilities by allowing addition of custom data fields to the underlying domain objects.

Basic steps in Domain Object extensions are:

Create new customized domain Object which will extend existing one and add additional data .



For example:

```

package com.ofss.digx.domain.collaboration.entity.customdemo;

import com.ofss.digx.domain.collaboration.entity.Collaboration;

/**
 * custom domain objet to handle the changes ate UI or pre hoot level
 */
public class CustomCollaborationDomainObject extends Collaboration{

    /**
     */
    private static final long serialVersionUID = 1L;

    /**
     * store the mobile number to call on.
     */
    private String mobileCustomer;

    /**
     * @return the mobileCustomer
     */
    public String getMobileCustomer() {
        return mobileCustomer;
    }

    /**
     * @param mobileCustomer the mobileCustomer to set
     */
    public void setmobileCustomer(String mobileCustomer) {
        this.mobileCustomer = mobileCustomer;
    }
}

```

Configure Customized domain object in database

The domain object created needs to be mapped as a custom domain object for the existing domain object. For example:

```
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG,
OBJECT_VERSION_NUMBER)
```

```
values ('com.ofss.digx.domain.origination.entity.submission.lending.application',
'CustomizedAbstractDomainObjectConfig',
'com.ofss.digx.domain.origination.entity.submission.lending.application.ext.Application', 'N', 'asdf',
'asdf', 'asdf', ", 'asdf', ", 'Y', 1);
```

Three main columns that need to be feed with new information are.

- CATEGORY_ID : “CustomizedAbstractDomainObjectConfig”
- PROP_VALUE:” CLASS NAME of the class implementing the custom domain object ”
- PROP_ID:” CLASS NAME OF THE DomainObject”.

Create Eclipse Mapping - You will need to create eclipse mapping to map the database table to the domain object. Follow these steps:

Create new ORM file to handle Customized Domain Object. This ORM file should contain entries for all columns in corresponding domain object table.

Add an entry for this ORM XML in the mapping configuration xml - eclipse configuration XML.

Create new table corresponds to newly created Domain Object.

Newly created ORM file will look like (CollaborationDemo.orm.xml):

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-mappings version="2.5" xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm" xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.eclipse.org/eclipselink/xsds/persistence/orm
  http://www.eclipse.org/eclipselink/xsds/eclipselink_orm_2_5.xsd">
3   <entity name="Collaboration" class="com.ofss.digx.domain.collaboration.entity.customdemo.CustomCollaborationDomainObject">
4     <table name="DIGX_CO_COLLABORATION" />
5     <attributes>
6       <embedded-id attribute-type="com.ofss.digx.domain.collaboration.entity.CollaborationKey" name="collaborationKey">
7         <attribute-override name="id">
8           <column name="ID" />
9         </attribute-override>
10      </embedded-id>
11      <basic attribute-type="java.lang.Integer" name="noOfParticipants">
12        <column name="NO_OF_PARTICIPANTS" />
13      </basic>
14      <basic attribute-type="java.lang.Integer" name="mobileCustomer"> <!-- Added for customDomain Object -->
15        <column name="MOBILE_INFO" />
16      </basic>
17      <basic attribute-type="java.lang.String" name="externalRefId">
18        <column name="EXTERNAL_REF_ID" />
19      </basic>
20      <basic attribute-type="java.lang.String" name="authorId">
21        <column name="AUTHOR_ID" />
22      </basic>
23      <basic attribute-type="java.lang.String" name="authorName">
24        <column name="AUTHOR_NAME" />
25      </basic>
26      <basic attribute-type="java.lang.String" name="collaborationType">
27        <column name="COLLABORATION_TYPE" />
28      </basic>
29      <basic attribute-type="java.lang.String" name="securityCode">
30        <column name="SECURITY_CODE" />
31      </basic>
32      <basic attribute-type="java.lang.String" name="url">
33        <column name="URL" />
34      </basic>
35      <basic attribute-type="com.ofss.fc.datatype.Date" name="expiryDate">
36        <column name="EXPIRY_DATE"/>
37        <convert>DateTime</convert>
38      </basic>
39      <basic attribute-type="java.lang.Boolean" name="status">
40        <column name="COLLABORATION_STATUS" />
41        <convert>yesno</convert>
42      </basic>
43      <one-to-many attribute-type="java.util.List" fetch="LAZY"
44        name="participantsList" target-entity="com.ofss.digx.domain.collaboration.entity.Participant">
45        <cascade>
46          <cascade-all />
47        </cascade>
48        <join-column insertable="false" name="COLLABORATION_ID" referenced-column-name="ID" updatable="true"/>
49      </one-to-many>
50      <basic attribute-type="com.ofss.digx.enumeration.collaboration.CollaborationChannel" name="collaborationChannel">
51        <column name="COLLABORATION_CHANNEL" />
52        <enumerated>VAIHTER</enumerated>

```

Extensible Markup Language file length: 3472 lines: 77 Ln:16 Col:21 Sel:155 Dos/Windows ANSI as UTF-8 INS

Mapping configuration xml(persistence.xml):

```

1  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
2  <persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0" xsi:schemaLocation=
3  "http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
4  <persistence-unit name="DIGX" transaction-type="RESOURCE_LOCAL">
5  <jta-data-source>DIGX</jta-data-source>
6  <mapping-file>META-INF/generic-mapping.orm.xml</mapping-file>
7  <mapping-file>orm/eclipselink/mappings/channels/Channel.orm.xml</mapping-file>
8  <mapping-file>orm/eclipselink/mappings/channels/channels-embeddable.orm.xml</mapping-file>
9  <mapping-file>orm/eclipselink/mappings/commonsms/sms.MetadataDefinition.orm.xml</mapping-file>
10 <mapping-file>orm/eclipselink/mappings/commonsms/SecureService.orm.xml</mapping-file>
11 <mapping-file>orm/eclipselink/mappings/commonsms/TransactionBlackout.orm.xml</mapping-file>
12 <mapping-file>orm/eclipselink/mappings/commonsms/commonsms.Query.orm.xml</mapping-file>
13 <mapping-file>orm/eclipselink/mappings/commonsms/commonsms-embeddable.orm.xml</mapping-file>
14 <mapping-file>orm/eclipselink/mappings/commonservices/Maintenances/BankParameters.orm.xml</mapping-file>
15 <mapping-file>orm/eclipselink/mappings/commonservices/Maintenances/BranchDatesDefinition.orm.xml</mapping-file>
16 <mapping-file>orm/eclipselink/mappings/commonservices/commonservices-embeddable.orm.xml</mapping-file>
17 <mapping-file>orm/eclipselink/mappings/core/core.Query.orm.xml</mapping-file>
18 <mapping-file>orm/eclipselink/mappings/core/TransactionReference.orm.xml</mapping-file>
19 <mapping-file>orm/eclipselink/mappings/core/core-embeddable.orm.xml</mapping-file>
20 <mapping-file>orm/eclipselink/mappings/mapper/keymapper.orm.xml</mapping-file>
21 <mapping-file>orm/eclipselink/mappings/mapper/keymapper-Query.orm.xml</mapping-file>
22 <mapping-file>orm/eclipselink/mappings/me/QueriesForMe.orm.xml</mapping-file>
23 <mapping-file>orm/eclipselink/mappings/party/PartyDetails.orm.xml</mapping-file>
24 <mapping-file>orm/eclipselink/mappings/session/nonce/session-nonce.orm.xml</mapping-file>
25 <mapping-file>orm/eclipselink/mappings/idgeneration/IdGenerationConfiguration.orm.xml</mapping-file>
26 <mapping-file>orm/eclipselink/mappings/idgeneration/commonservices-embeddable.orm.xml</mapping-file>
27 <mapping-file>orm/eclipselink/mappings/idgeneration/core.Query.orm.xml</mapping-file>
28 <mapping-file>orm/eclipselink/mappings/collaboration/CollaborationDemo.orm.xml</mapping-file> <!-- mapping for customDomain Object -->
29 <mapping-file>orm/eclipselink/mappings/collaboration/Collaboration-embeddable.orm.xml</mapping-file>
30 <mapping-file>orm/eclipselink/mappings/collaboration/Collaboration-Query.orm.xml</mapping-file>
31 <mapping-file>orm/eclipselink/mappings/collaboration/participant/Participant.orm.xml</mapping-file>
32 <mapping-file>orm/eclipselink/mappings/collaboration/participant/Participant-embeddable.orm.xml</mapping-file>
33 <mapping-file>orm/eclipselink/mappings/collaboration/participant/Participant-Query.orm.xml</mapping-file>
34
35 <properties>
36 <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
37 <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@ofss310416.in.oracle.com:1521:obcp"/>
38 <property name="javax.persistence.jdbc.user" value="ODP_1501"/>
39 <property name="javax.persistence.jdbc.password" value="odpadmin"/>
40 <!--property name="eclipselink.ddl-generation" value="create-tables"/-->
41 <property name="eclipselink.logging.level.sql" value="FINE"/>
42 <property name="eclipselink.jdbc.connector" value="com.offss.fc.infra.das.orm.eclipselink.EclipseLinkConnector"/>
43
44 </properties>
45 </persistence-unit>
46 </persistence>

```

Here Assembler should fetch customized domain object. Following example shows Assembler calls `getCustomizedDomainObject` which returns customized domain object with mapping of `nameValuePairDTOArray` to this customized domain Object internally.

For example:

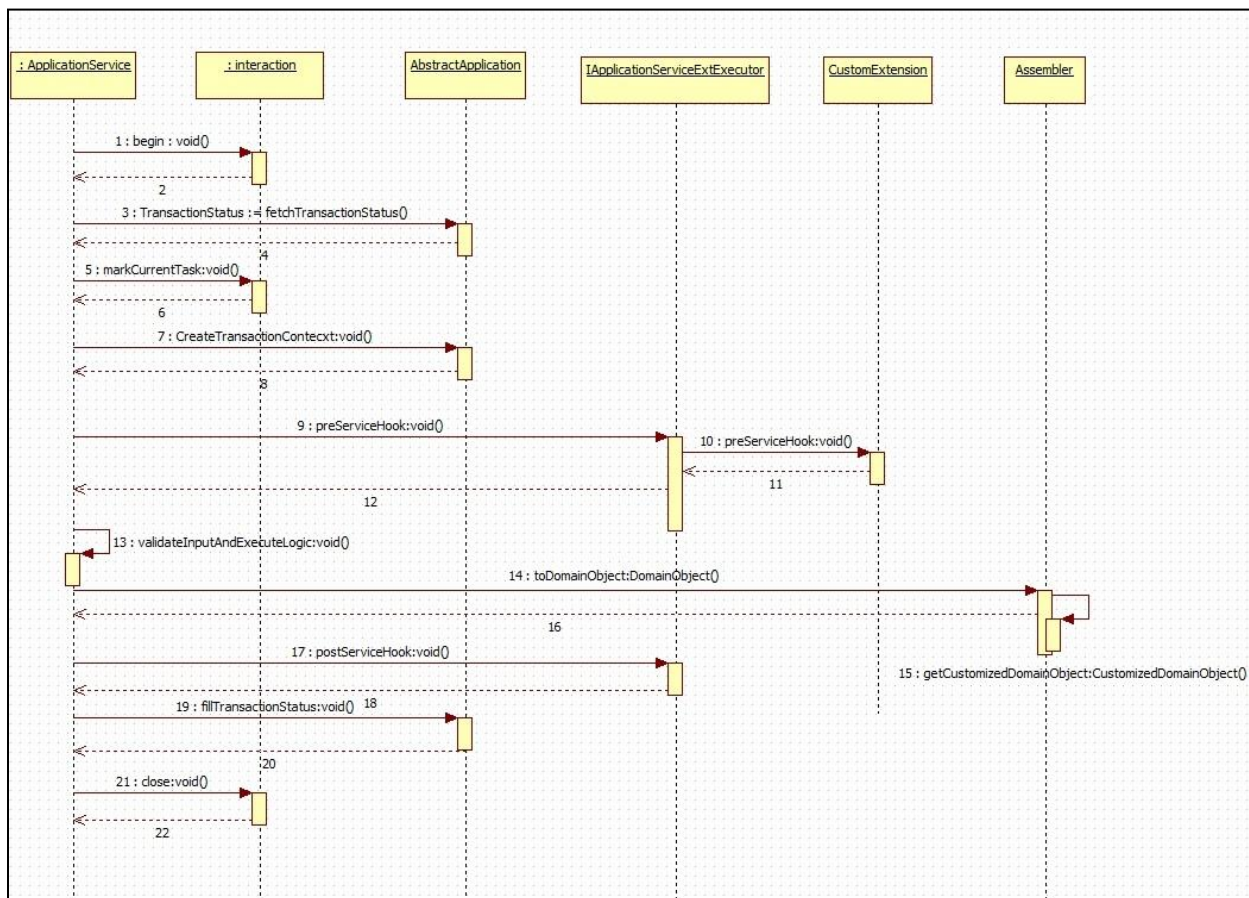
```

7
public Collaboration toDomainObject(CollaborationDTO collaborationDTO) {
    Collaboration collaboration;
    if (collaborationDTO.getDictionaryArray() != null) {
        try {
            collaboration = (Collaboration) getCustomizedDomainObject(collaborationDTO);
        } catch (java.lang.Exception e) {
            collaboration = new Collaboration();
        }
    } else {
        collaboration = new Collaboration();
    }

    CollaborationKey collaborationKey = new CollaborationKey();
    if (collaborationDTO.getId() != null) {
        collaborationKey.setId(collaborationDTO.getId());
    }
    collaboration.setExternalRefId(collaborationDTO.getExternalRefId());
    collaboration.setAuthorId(collaborationDTO.getAuthorId());
    collaboration.setAuthorName(collaborationDTO.getAuthorName());
    collaboration.setUrl(collaborationDTO.getUrl());
    collaboration.setExpiryDate(collaborationDTO.getExpiryDate());
    collaboration.setNoOfParticipants(collaborationDTO.getNoOfParticipants());
    collaboration.setCollaborationChannel(collaborationDTO.getCollaborationChannel());
    List<Participant> participantsList = new ArrayList<Participant>();
    if (collaborationDTO.getParticipants() != null && !collaborationDTO.getParticipants().isEmpty()) {
        for (ParticipantDTO participantDTO : collaborationDTO.getParticipants()) {
            Participant participant = new Participant();
            ParticipantKey participantKey = new ParticipantKey();
            participant.setKey(participantKey);
            participant.setInternalPartyId(participantDTO.getInternalPartyId());
            participant.setName(participantDTO.getName());
            participant.setPublishingId(participantDTO.getPublishingId());
            participant.setIsPublished(participantDTO.getIsPublished());
            participantsList.add(participant);
        }
        collaboration.setParticipants(participantsList);
    }
    collaboration.setCollaborationKey(collaborationKey);
    return collaboration;
}

```

3.3 Sequence Diagram



3.3.1 Adapter Mechanism

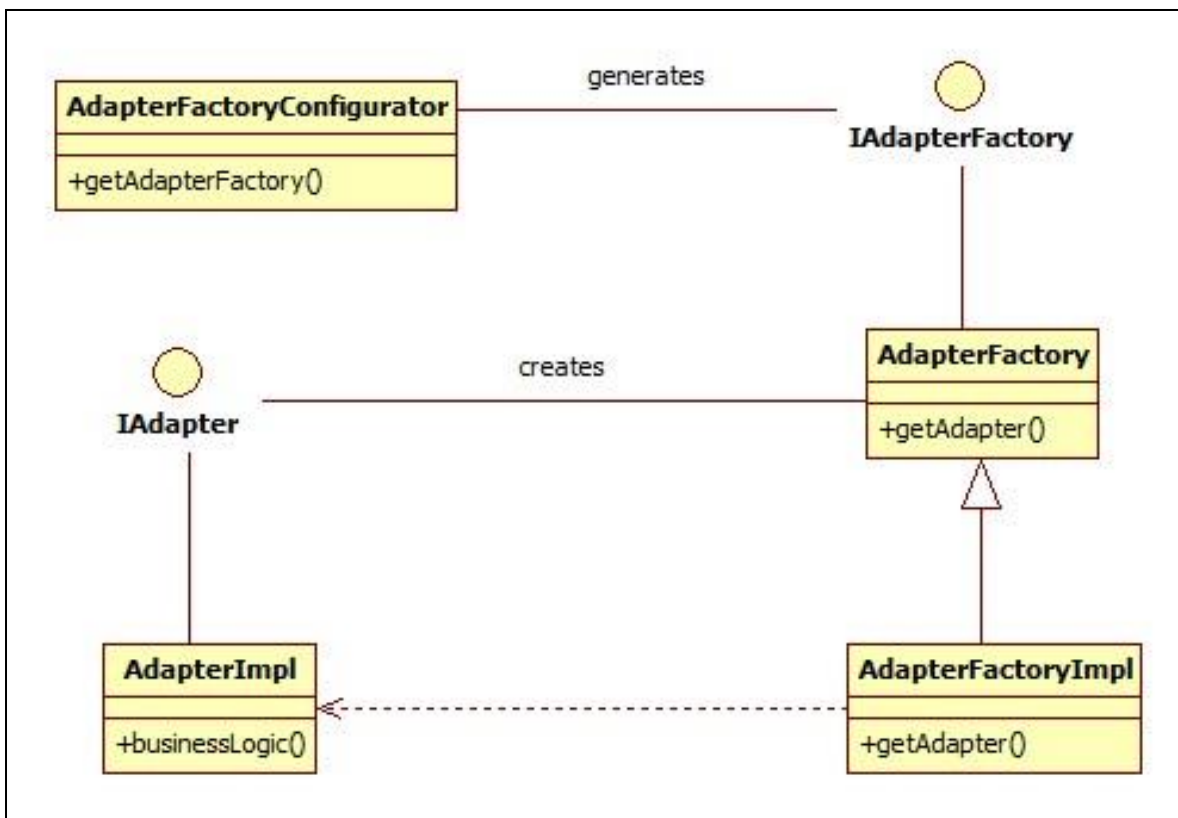
An adapter, by definition, helps the interfacing or integrating components adapt. In software it represents a coding discipline that helps two different modules or systems to communicate with each other and helps the consuming side adapt to any incompatibility of the invoked interface work together. Incompatibility could be in the form of input data elements which the consumer does not have and hence might require defaulting or the invoked interface might be a third party interface with a different message format requiring message translation. Such functions, which do not form part of the consumer functionality, can be implemented in the adapter layer.

3.4 Adapter Mechanism Class Diagram

An Application Service in calling module calls the **getAdapterFactory()** method of class *AdapterFactoryConfigurator* which returns an instance of an implementation of the abstract class *AdapterFactory*. The class of instance is decided by the string parameter passed to the method.

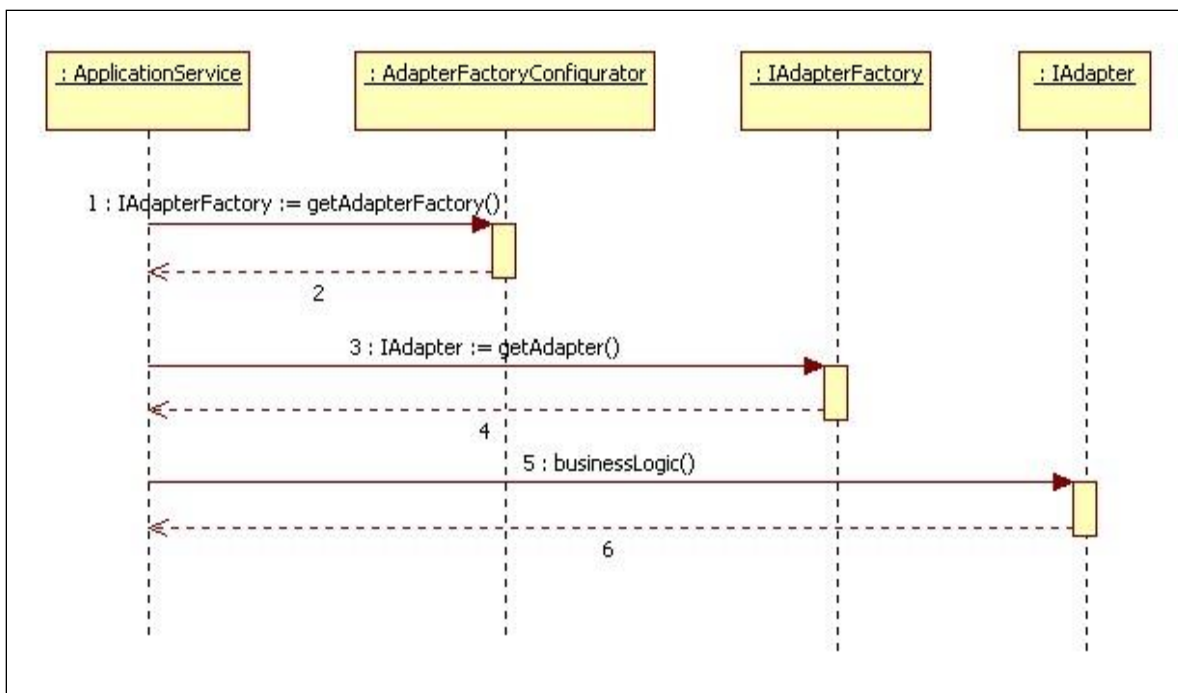
The **getAdapter()** method in the AdapterFactory returns an adapter instance. The class of instance is decided by the string parameter passed to the method.

The Application Service then uses this adapter instance to access any data from an application service within another module.



3.5 Adapter Mechanism Sequence Diagram

A method in an application service gets an instance of a desired adapter factory by calling **getAdapterFactory()** method of *AdapterFactoryConfigurator* class. The instance of the adapter factory obtained is used to call **getAdapter()** method to get an instance of the adapter. This adapter instance has all the methods to communicate to the service in another module.



3.6 Customizing Existing Adapters (Custom Adapter)

If an added functionality or replacement functionality is required for an existing adapter or existing method in an adapter, the customization developer has to develop a new adapter and corresponding adapter factory and override the method in a new custom adapter class. The custom adapter would have to override and implement the methods which need changes.

```

2* Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.
4 package com.ofss.fc.app.adapter.party;
5
6 import java.util.List;
7
8 import com.ofss.fc.app.context.SessionContext;
9 import com.ofss.fc.app.loan.dto.LoanBalanceInquiryResponse;
10 import com.ofss.fc.app.loan.dto.account.unified.inquiry.LoanAccountUnifiedInquiryResponse;
11 import com.ofss.fc.app.loan.dto.disbursement.loghistory.LoanDisbursementLogHistoryResponse;
12 import com.ofss.fc.app.party.dto.LoanAccountAttributesDTO;
13 import com.ofss.fc.app.party.dto.relation.account.preferences.ChannelFacilitiesDTO;
14 import com.ofss.fc.enumeration.loan.LoanAccountStatusType;
15 import com.ofss.fc.framework.context.ApplicationContext;
16 import com.ofss.fc.infra.exception.FatalException;
17
18 /**
19  * This class serves as the adapter class for the Party and loans Integration mainly for the Single Party View<br>
20  *
21  * @author VallabhM
22  */
23 public interface ICustomerLoansAdapter {
24
25     /**
26      * Service to return all the loans Account information to be displayed in SPIV<br>
27      *
28      * @param applicationContext
29      * @param partyId
30      * @return LoanAccountAttributesDTO
31      */
32     public abstract LoanAccountAttributesDTO[] fetchLoanAccountsInformation(ApplicationContext applicationContext, String partyId);
33
34     public abstract void fetchLNInformation(String accountId);
35
36     /**
37      * Maintain primary Account Holder ID for Loan Account.
38      *
39      * @param accountId
40      * @param newPartyId
41      */
42     public void primaryAccountHolderIDMaintenance(String accountId, String newPartyId);
43
44     /**
45      * @param loanAccounts
46      * @param LoanAccountStatusType
47      */
48     public abstract void modifyStatusOfLoanAccounts(List<String> loanAccounts, LoanAccountStatusType LoanAccountStatusType);
49
50     /**
51      * Method to fetch Outstanding, RPA and unclear balances for a given loan account
52      * @param accountId
53      * @throws FatalException
54      */
55     public abstract LoanBalanceInquiryResponse inquireLoanBalance(ApplicationContext applicationContext, String accountId) throws FatalException;
56
57

```

3.6.1 Custom Adapter Example

We take the example of `LoanApplicationRequirementAdapter`. For example the requirement is to send an email alert when the requirements of a particular loan application are updated. The clip application by default does not provide any integration with an SMTP/Email server. The additional interfacing with the gateway can be done in the custom adapter. . The following steps would have to be followed for implementation of a custom `LoanApplicationRequirementAdapter`.

Develop a *CustomLoanApplicationRequirementAdapter* and *Custom LoanApplicationRequirementAdapterFactory*. As a guideline, the custom adapter should extend the existing adapter and override the methods which needs to be replaced with new functionality.

For Example:

```

29     logger.log(
30         Level.SEVERE,
31         formatter
32             .formatMessage(
33                 "Remote Webservice call Failed while "
34                 + "updating loan requirements for submission id: %s in update method of LoanApplicationRequirementAdapter",
35                 submissionId);
36     }
37     responseHandler.translateAndThrow(e, this.getClass());
38 }
39 } catch (FatalException e) {
40     if (logger.isLoggable(Level.SEVERE)) {
41         logger.log(
42             Level.SEVERE,
43             formatter
44                 .formatMessage(
45                 "Failed while "
46                 + "updating loan requirements for submission id: %s in update method of LoanApplicationRequirementAdapter",
47                 submissionId);
48         logger.log(Level.SEVERE, formatter.formatMessage("Error: ", e));
49     }
50     responseHandler.exceptionForUpdate(e, e.getFaultInfo());
51 }
52 responseHandler.fillTransactionStatus(response.getStatus());
53 loanApplicationRequirement.setFacilityId(response.getFacilityId());
54 loanApplicationRequirement.setProductGroupSerialNumber(response.getProductGroupSerialNumber());
55 loanUpdateResponse.setLoanApplicationRequirementDTO(loanApplicationRequirement);
56
57 // Your code for the SMTP mail client invocation goes here.
58 Properties props = new Properties();
59 props.setProperty("mail.smtp.host", "smtp.example.com");
60
61 Session session = Session.getInstance(props, null);
62 Transport transport = session.getTransport("smtp");
63 transport.connect("user", "password");
64
65 Message message = new MimeMessage(session);
66 message.setSubject("Test");
67 message.setText("Hello :)");
68 message.setFrom(new InternetAddress("you@example.com"));
69 message.setRecipient(Message.RecipientType.TO, new InternetAddress("your-friend@example.com"));
70 transport.sendMessage(message, message.getAllRecipients());
71
72 if (logger.isLoggable(Level.FINE)) {
73     logger.log(Level.FINE, formatter.formatMessage(
74         "Exiting from method create of LoanApplicationRequirementAdapter loanCreationResponse = %s",
75         loanUpdateResponse));
76 }
77 return loanUpdateResponse;
78 }
79
80 /**
81  * Reads the loan requirement set by the party from the application resource. Returns the instance of loan
82  * requirement that is set by the party. Fetches all lending product details and return first lending product ,if

```

3.6.2 Custom Adapter Configuration.

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values ('IS_LOAN_APPLICATION_REQUIREMNT_ADAPTER_CUSTOM', 'customadapterconfig', 'true', 'N', 'asdf', 'asdf', 'asdf', '', 'asdf', '', 'Y', 1);

3.7 Mock Adapter

Mock adapter represents the intermediate layer required for communicating with third party host system. operations supported by Mock adapter are create, fetch, etc. Mock Adapter is basically responsible for generating required response from the core banking system for a given request.

3.7.1 Mock Adapter Example.

Here it is responsible for communicating with third party host system as part of Loan requirement for a party.

```

LoanApplicationRequirementMockAdapter.java
1 package com.ofss.digx.app.origination.adapter.impl.submission.application;
2
3 import java.util.logging.Level;[]
15
16 /**
17  * Mock adapter represents the intermediate layer required for communicating with third party host system as part of
18  * Loan requirement for a party.<br/>
19  * Following operations are supported by this adapter.
20  * <ol>
21  * <li>Create loan application based on submission Identifier and loan details</li>
22  * <li>Fetch loan application based on submission Identifier and facility Identifier</li>
23  * </ol>
24  */
25 public class LoanApplicationRequirementMockAdapter implements ILoanApplicationRequirementAdapter {
26
27     /**
28      * The component name for this class.
29      */
30     private static final String THIS_COMPONENT_NAME = LoanApplicationRequirementMockAdapter.class.getName();
31
32     /**
33      * Instance of {@link MultiEntityLogger}.
34      */
35     private static final MultiEntityLogger formatter = MultiEntityLogger.getUniqueInstance();
36
37     /**
38      * Instance of {@link java.util.logging.Logger} to support multi-entity wide logging.
39      */
40     private static final Logger logger = formatter.getLogger(THIS_COMPONENT_NAME);
41
42     /**
43      * The Simple name for this class.
44      */
45     private static final String THIS_SIMPLE_NAME = LoanApplicationRequirementMockAdapter.class.getSimpleName();
46
47     /**
48      * Creates the loan requirements of a party. An application for the loan requirement is created using submission
49      * identifier and {@link LoanApplicationRequirementDTO} containing requested Amount, requested Tenor, list of
50      * variants, etc. Returns the response containing information of successful or unsuccessful creation of the loan
51      * requirement.
52      *
53      * @param submissionId
54      *        unique identifier of the submitted application
55      * @param loanApplicationRequirement
56      *        {@link LoanApplicationRequirementDTO} representing the loan requirement submitted by the applicant
57      * @return {@link LoanApplicationCreateResponseDTO} referring the response generated post creation of the loan
58      *         requirement.
59      * @throws Exception
60      *         if the core banking system is not able to create a loan requirement based on details provided.
61      */
62     @Override
63     public LoanApplicationCreateResponseDTO create(String submissionId,
64         LoanApplicationRequirementDTO loanApplicationRequirement) throws Exception {
65

```

3.7.2 Mock Adapter Configuration:Database needs to be configured for the execution of the Mock Adapter corresponding to a service.

insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER)

values ('PARTY_COLLECTION_ADAPTER MOCKED', 'adapterfactoryconfig', 'true', 'N', 'asdf', 'asdf', 'asdf', ", 'asdf', ", 'Y', 1);

4. Outbound Webservice Extension.

The outbound webservice configurations are set of properties defined to invoke services from the host. The host is the core bank system where the business logic for core banking facilities is written and contains the corresponding services to access that data. The existing CLIP application has an Adapter layer which directly interacts with the host. There are extension endpoints available for configuring a different host in the adapter layer. Following steps need to be followed:

4.1 Using your own Web Service constants

The web service constants will change depending on the WSDL specification provided by the host system. An Example WebServiceConstants file is shown below:

```

1 package com.ofss.digx.common;
2
3 /**
4  * Constants for web service invocation from the adapter implementation.
5  */
6 public class WebserviceConstants {
7
8     /**
9      * Holds the service name to be invoked from the adapter.
10     */
11
12     public static final String PRODUCT_MANUFACTURING_APPLICATION_SERVICE = "ProductManufacturingApplicationServiceSpi";
13
14     /**
15      * Holds the Offer Inquiry Application service name to be invoked from the adapter.
16     */
17     public static final String OFFER_INQUIRY_APPLICATION_SERVICE_SPI = "OfferInquiryApplicationServiceSpi";
18     /**
19      * Holds the Purpose Application Service Spi name to be invoked from the adapter.
20     */
21     public static final String PURPOSE_APPLICATION_SERVICE_SPI = "PurposeApplicationServiceSpi";
22     /**
23      * Holds the Submission Creation Application Service Spi name to be invoked from the adapter.
24     */
25     public static final String SUBMISSION_CREATION_APPLICATION_SERVICE_SPI = "SubmissionCreationApplicationServiceSpi";
26
27     /**
28      * Holds the Submission Product Application Service Spi name to be invoked from the adapter.
29     */
30     public static final String SUBMISSION_PRODUCT_APPLICATION_SERVICE_SPI = "SubmissionProductApplicationServiceSpi";
31
32     /**
33      * Holds the Detailed Application Tracker Application Service Spi name to be invoked from the adapter.
34     */
35     public static final String DETAILED_APPLICATION_TRACKER_APPLICATION_SERVICE_SPI = "DetailedApplicationTrackerApplicationServiceSpi";
36
37     /**
38      * Holds the operation name to fetch list of all product groups.
39     */
40     public static final String FETCH_ALL_PRODUCT_GROUPS = "fetchAllProductGroups";
41
42     /**
43      * Holds the operation name to fetch list of all products for the group code.
44     */
45     public static final String FETCH_ALL_PRODUCTS_FOR_GROUP_CODE = "fetchAllProductsForGroupCode";
46
47     /**
48      * Holds the method name of host to fetch offers linked to product.
49     */
50     public static final String FETCH_OFFERS_LINKED_TO_PRODUCT = "fetchOffersLinkedToProduct";
51     /**
52      * Holds the method name of host to fetch purpose code linked to a group code.
53     */
54     public static final String FETCH_PURPOSE_CODES_FOR_GROUP_CODE = "fetchPurposeCodesForGroupCode";

```


4.2 Making the Database Entries

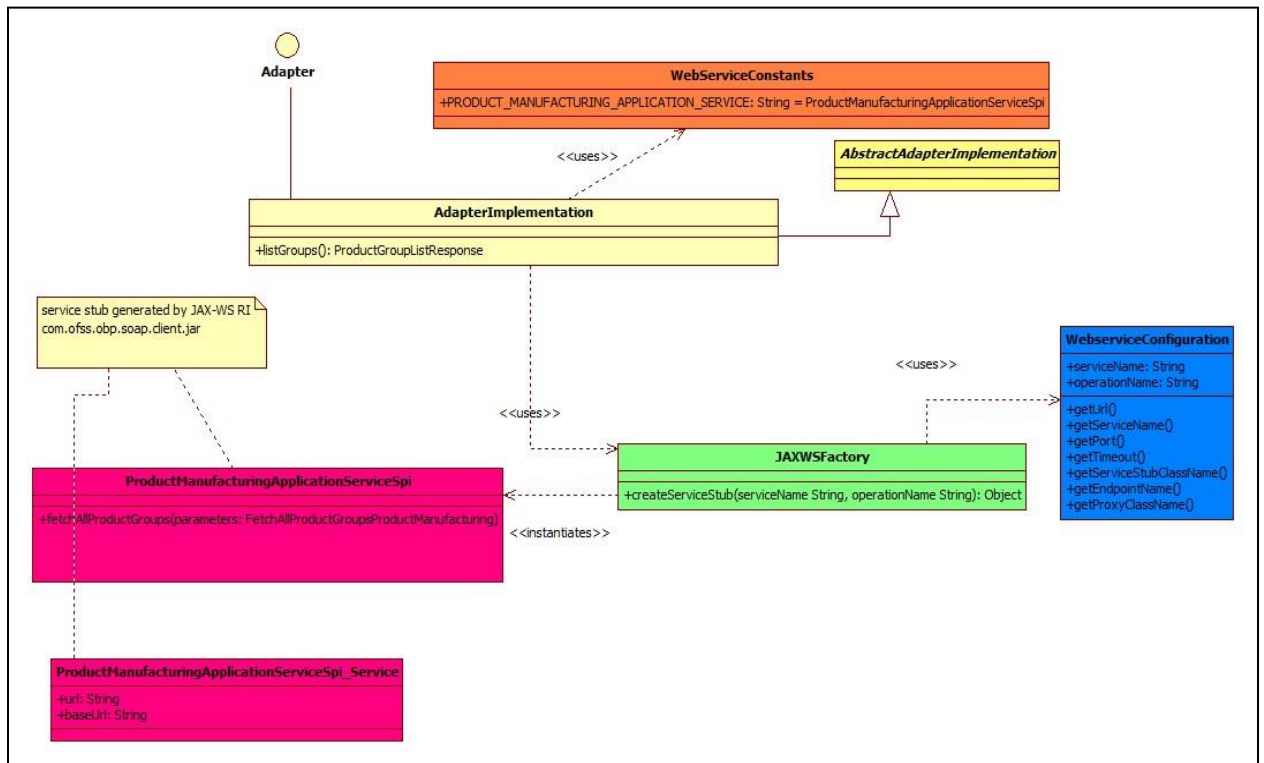
digx_fw_config_out_ws_cfg_b. Holds the entries for the host service endpoints.

For Example:

```
insert into digx_fw_config_out_ws_cfg_b (SERVICE_ID, PROCESS, URL, ENDPOINT_URL,
NAMESPACE, TIME_OUT, SERVICE, STUB_CLASS, SECURITY_POLICY, ENDPOINT_NAME,
STUB_SERVICE, HTTP_BASIC_AUTH_CONNECTOR, HTTP_BASIC_AUTH_REALM,
PROXY_CLASS_NAME, IP, PORT, USERNAME, PASSWORD, CREATED_BY,
LAST_UPDATED_BY, CREATION_DATE, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER, ANONYMOUS_SECURITY_POLICY,
ANONYMOUS_SECURITY_KEY_NAME)
```

```
values ('inquireApplication', 'BaseApplicationServiceSpi',
'http://ofss310406.in.oracle.com:8001/com.ofss.fc.webservice/services/origination/BaseApplicationServiceSpi?wsdl', '',
'http://application.core.service.origination.appx.fc.ofss.com/BaseApplicationServiceSpi', 1200000,
'BaseApplicationServiceSpi', '', 'BaseApplicationServiceSpiPort',
'com.ofss.fc.appx.origination.service.core.application.baseapplicationservice
```

4.3 Class Diagram of Components Involved



4.4 Client Jar

Generate the corresponding service stubs from the WSDL specifications using The JAX-WS RI tool. Package the generated code as a jar and include it in the Adapter implementation.

4.5 Custom Adapter

Lastly create a custom adapter to handle the changes made in the host configurations. The custom adapter will be using the JAXWSFactory to create instances of the desired service stubs. The rest of the custom adapter implementation is the same as mentioned in the section 5.3

For example:

```

99 duration.setDays(loanApplicationRequirement.getRequestedTenure().getDays());
100 duration.setMonths(loanApplicationRequirement.getRequestedTenure().getMonths());
101 duration.setYears(loanApplicationRequirement.getRequestedTenure().getYears());
102
103 loanProduct.setRequestedAmount(money);
104 loanProduct.setRequestedTenor(duration);
105 loanProduct.setPurpose(loanApplicationRequirement.getPurpose());
106 loanProduct.setPurposeType(PurposeTypes.valueOf(loanApplicationRequirement.getPurposeType()));
107 loanProduct.setIsCapitalizeFeesOpted(loanApplicationRequirement.getIsCapitalizeFeesOpted());
108 loanProduct.setIsSettlementRequired(loanApplicationRequirement.getIsSettlementRequired());
109 loanProduct.setProductGroupId(loanApplicationRequirement.getProductGroupId());
110 loanProduct.setProductGroupName(loanApplicationRequirement.getProductGroupName());
111
112 AddNewLoanProductSubmissionProduct newProduct = new AddNewLoanProductSubmissionProduct();
113 newProduct.setSessionContext(AdapterContextHelper.getInstance().setContext());
114 newProduct.setSubmissionId(submissionId);
115 newProduct.setLoanProductDTO(loanProduct);
116 LoanProductResponse response = null;
117
118 SubmissionProductApplicationServiceSpi clientProcess = null;
119 try {
120     clientProcess = (SubmissionProductApplicationServiceSpi) JAXWSFactory.createServiceStub(
121         WebserviceConstants.SUBMISSION_PRODUCT_APPLICATION_SERVICE_SPI,
122         WebserviceConstants.ADD_NEW_LOAN_PRODUCT);
123     response = clientProcess.addNewLoanProduct(newProduct).getReturn();
124 } catch (WebServiceException e) {
125     if (Logger.isLoggable(Level.SEVERE)) {
126         Logger.log(
127             Level.SEVERE,
128             formatter
129                 .formatMessage(
130                     "Remote Webservice call Failed while "
131                     + "creating loan requirements for submission id: %s in create method of LoanApplicationRequirementAdapter",
132                     submissionId));
133     }
134     responseHandler.translateAndThrow(e, this.getClass());
135 }
136 } catch (FatalException e) {
137     if (Logger.isLoggable(Level.SEVERE)) {
138         Logger.log(
139             Level.SEVERE,
140             formatter
141                 .formatMessage(
142                     "Failed while "
143                     + "creating loan requirements for submission id: %s in create method of LoanApplicationRequirementAdapter",
144                     submissionId));
145         Logger.log(Level.SEVERE, formatter.formatMessage("Error: ", e));
146     }
147     responseHandler.exceptionForCreate(e, e.getFaultInfo());
148 }
149 responseHandler.fillTransactionStatus(response.getStatus());
150 loanCreationResponse.setSubmissionId(submissionId);
151 loanApplicationRequirement.setFacilityId(response.getFacilityId());
152 loanApplicationRequirement.setProductGroupSerialNumber(response.getProductGroupSerialNumber());

```

5. Security Customizations

CLIP comprising of several modules has to interface with various systems in an enterprise to transfer/share data which is generated during business activity that takes place during teller operations or processing. While managing the transactions that are within CLIP, it is needed to consider security & identity management and the uniform way in which these services need to be consumed by all applications in the enterprise.

This is possible if these capabilities can be externalized from the application itself and are implemented within products that are specialized to handle such services. Examples of these services include authentication against an enterprise identity-store, creating permissions and role based authorization model that controls access to not only the components of the application, but also the data that is visible to the user based on fine-grained entitlements.

5.1 Security Configuration

Method `checkAccess()` in class **AbstractSecureApplication** called from `ApplicationService` checks flag `IS_SECURITY_ENABLED`. If this flag has value `true` then it checks for security. Below image depicts same.

```
protected void checkAccess(String serviceId, Object... inputObjects) throws FatalException {
    if (logger.isLoggable(Level.FINE)) {
        logger.log(Level.FINE, formatter.formatMessage(ENTERED_INTRO + ".checkAccess. Service='%s'", serviceId));
    }
    logServiceCall(serviceId);
    this.currentService = serviceId;
    //This will get cleared automatically in final Interaction.close()
    if (!StringHelper.isNotNullCheckSpace((String) ThreadAttribute.get(ThreadAttribute.CURRENTLY_EXECUTING_SERVICE)))
        ThreadAttribute.set(ThreadAttribute.CURRENTLY_EXECUTING_SERVICE, serviceId);
    }
    boolean propertyValue = false;
    if (securityConstants == null) {
        securityConstants = ConfigurationFactory.getInstance().getConfigurations(SEcurityConstants);
    }
    String IS_COLLECTION_BATCH = (String) ThreadAttribute.get(ThreadAttribute.IS_COLLECTION_BATCH);
    propertyValue = securityConstants.getBoolean(IS_SECURITY_ENABLED, false);
    if (!propertyValue || (IS_COLLECTION_BATCH != null && "true".equalsIgnoreCase(IS_COLLECTION_BATCH))) {
        ThreadAttribute.set(ThreadAttribute.STR_ROLE, ANONYMOUS_ROLE);
        if (ThreadAttribute.get(ThreadAttribute.SMS) == null) {
            boolean isCreateWorkitemEnabled = false;
            if (securityConstants == null) {
                securityConstants = ConfigurationFactory.getInstance().getConfigurations(SEcurityConstants);
            }
            isCreateWorkitemEnabled = securityConstants.getBoolean(IS_CREATE_WORKITEMDITO_IN_DEV_ENABLED, true);
            if (isCreateWorkitemEnabled) {
                List<String> services = fetchAllOverriddenServices();
            }
        }
    }
}
```

Following Query is used to configure `IS_SECURITY_ENABLED` flag in database.

```
insert into digx_fw_config_all_b (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS_FLAG,
OBJECT_VERSION_NUMBER)
```

```
values ('IS_SECURITY_ENABLED', 'SecurityConstants', 'true', 'N', '', 'Security Constant',
'ofssuser', '30-JUN-15 04.04.49.000000 PM', 'ofssuser', '30-JUN-15 04.04.49.000000 PM', 'Y', 1);
```

`checkAccess()` calls method `checkAccessInternal()`. This method delegates the calls to the appropriate methods during security access check process. First of all, if the service is blacked out for the period in which the user is accessing it, then he will not be allowed to perform the transaction. Access policies are maintained on OID server corresponding to roles. Based on the policies, the user has access on a particular service or not is validated. If the service under

execution is eligible for OAAM validations, call is made to OAAM. OAAM validates the policies and rules set for the service and returns the action to be taken.

Action can be Allow, Block, Challenge1FA, Challenge1FADelay, Challenge2FA, Challenge2FADelay. Next **assertRequiredAuthorizations** is used to test for business conditions that can be either rejected, ignored or overridden. If status is marked as overridden, system fetches the configured workflow and distributes workitems accordingly.

checkAccessInternal has call to method assertAccessPolicies.

assertAccessPolicies is responsible to check whether user has access to a particular service or not.

5.2 Security Functions with the Extensibility Features

The following security functions are provided with the extensibility features:

5.2.1 Attributes participating in access policy rules

CLIP uses OES (Oracle Entitlement Server) to assert role-based access policies. Access policies are rules-based to give more flexibility.

Example of an access policy rule –

```
Grant

Role=CUSTOMERS_COM_OFSS_DIGX_APP_PARTY_SERVICE_CORE_EMPLOYMENTPROFILE_LIST_PERFORM_GRANT_PL

Service(Target)= com.ofss.digx.app.party.service.core.EmploymentProfile.list


Action = perform

Principals=Customer
```

Following Images shows steps to create new Authorization Rule

Step1.

- Click **New** Under Authorization Policies Tab

ORACLE Entitlements Server Accessibility Help Sign Out 
Signed in as weblogic

Authorization Management **System Configuration**

Home Search Authorization Poli... Untitled Untitled

Welcome to Authorization Policy Manager, the administration console for Oracle Entitlements Server.

Select an application to manage, then select an action from the links below.

Application Name	Resource Types	Role Mapping Policies
OAM11gApplication	New	New
OBP	Search	Search
CLIP		
b2bui		
soa-infra		

Application Roles	Resources	Entitlements
New	New	New
Search	Search	Search

Authorization Policies
New
Search

Note: objects will be created in the default domain

Help Center

- [Using Oracle Entitlements Server](#)
- [Navigating the Administration Console](#)
- [Understanding Application Roles](#)
- [Managing Authorization Policies](#)
- [Configuring Delegated Administrators](#)

Search and Create

- Search - External Roles
- Search - Users
- Search - Applications
- Create - Application

Copyright © 2009, 2013, Oracle and/or its affiliates. All rights reserved. Oracle Entitlements Server | 11.1.2.2.0

- Add Name and description details, Principles, Targets and Click **Save**.

The screenshot displays the Oracle Entitlements Server interface for configuring an authorization policy. The top navigation bar includes the Oracle logo, 'Entitlements Server', and user information: 'Signed in as weblogic'. The main interface is divided into a left-hand navigation pane and a central configuration area.

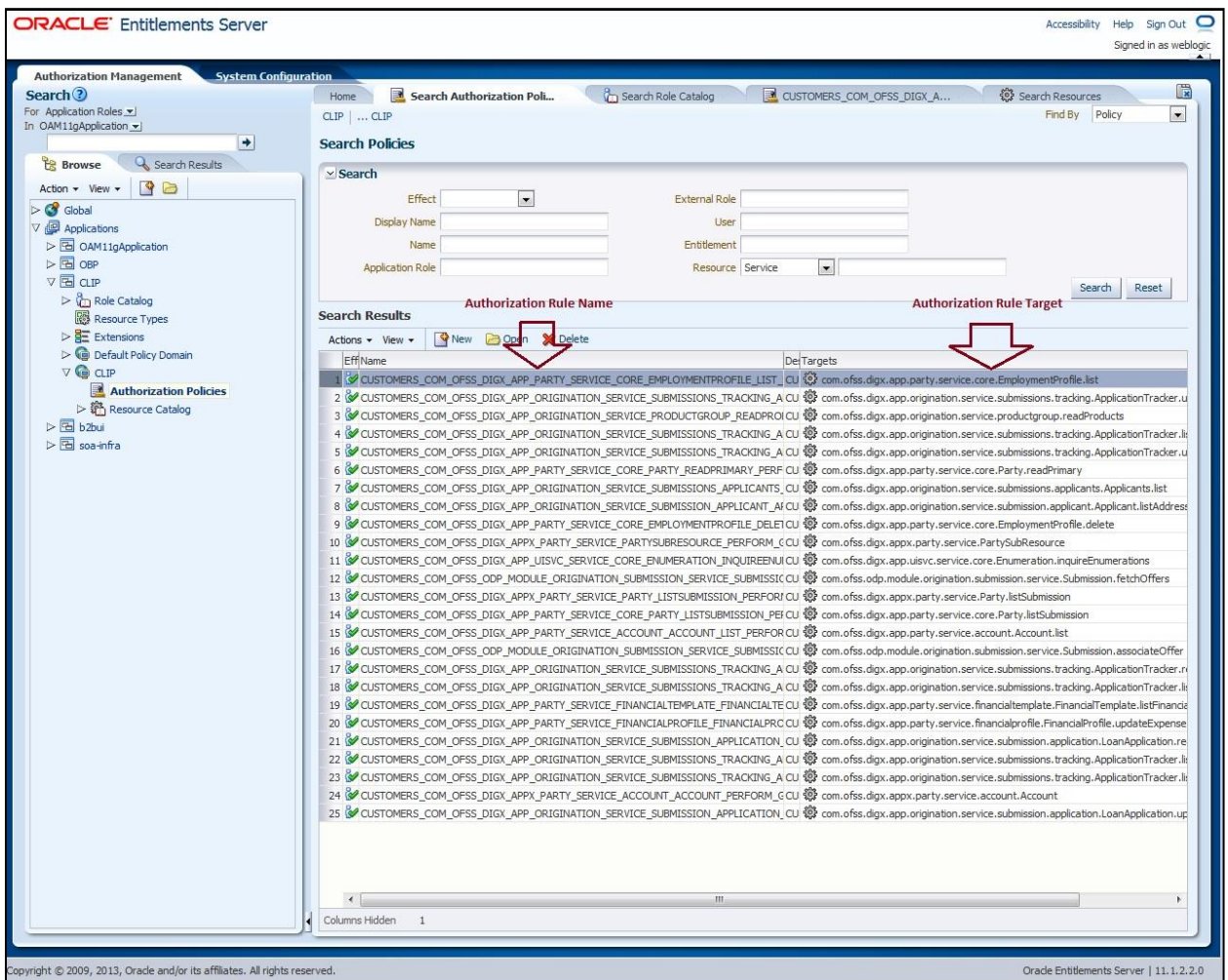
Navigation Pane: Shows a tree view under 'Authorization Policies' with sub-items like 'Global', 'Applications', 'OAM11gApplication', 'OBP', 'CLIP', 'Role Catalog', 'Resource Types', 'Extensions', 'Default Policy Domain', 'Resource Catalog', 'b2bui', and 'soa-infra'.

Configuration Area: Titled 'Untitled', it contains several sections:

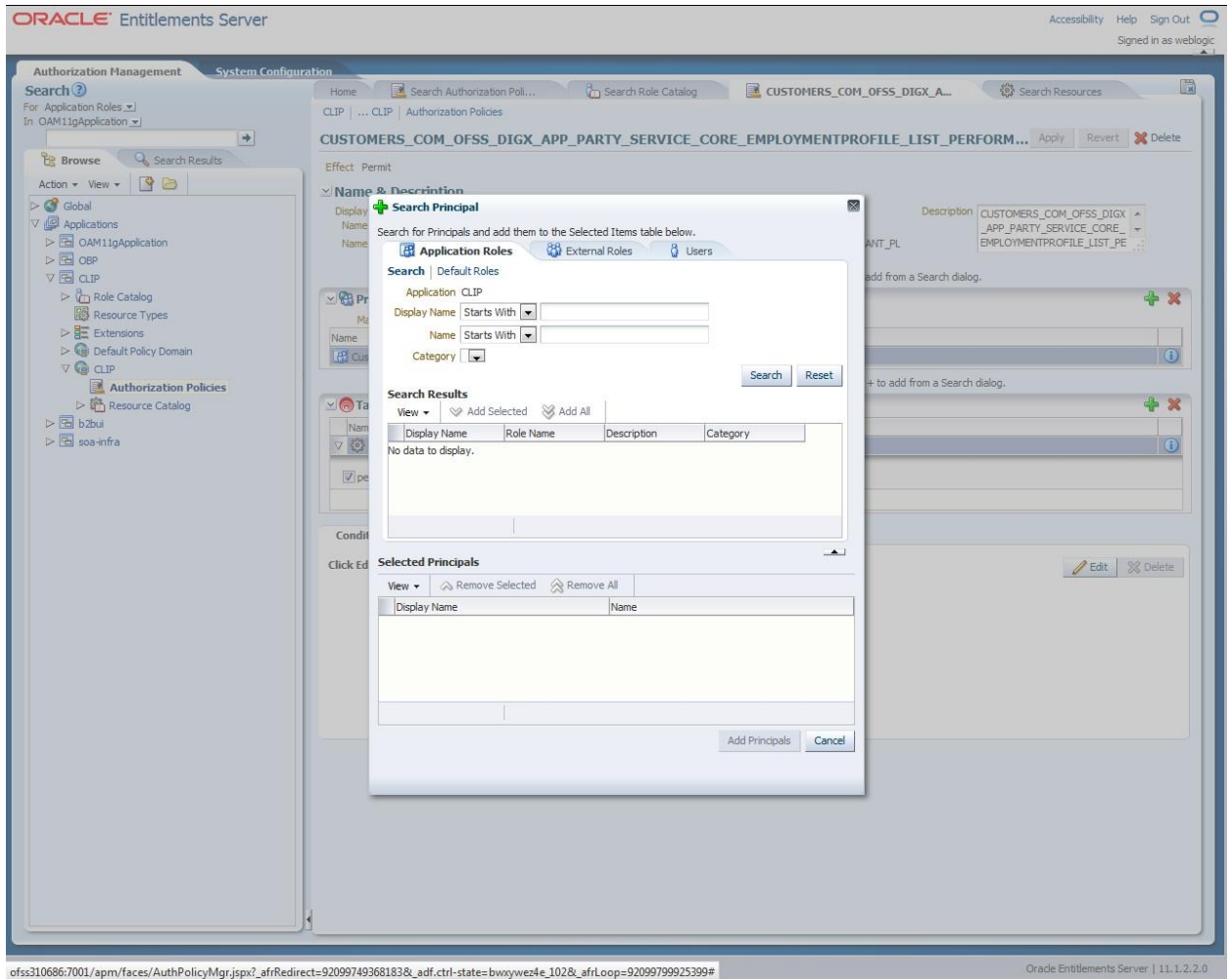
- Name & Description:** Includes fields for 'Display Name' and 'Description', and a '* Name' field.
- Principals:** A section for adding roles and users. It features a 'Match' dropdown (set to 'Any') and a table with 'Name' and 'Enabled Actions' columns. A red circle highlights the '+' icon for adding a principal.
- Targets:** A section for adding resources and entitlements. It features a table with 'Name' and 'Enabled Actions' columns. A red circle highlights the '+' icon for adding a target.
- Condition:** A section with a tab for 'Obligations' and a prompt to 'Click Edit Condition to build a new one'. It includes 'Edit' and 'Delete' buttons.

At the bottom of the page, there is a URL: 'ofss310686:7001/apm/faces/AuthPolicyMgr.jspx?_afrcRedirect=94338131209287&_adf.ctrl-state=bwoywez4e_471&_afrcLoop=94338238951714#' and the version 'Oracle Entitlements Server | 11.1.2.2.0'.

Following image shows example of access policy rule



The security framework OES(Oracle Entitlement Server) allows for addition to the facts that can be used in rules. Rule's Principles and Targets can be added. Following image shows screen to add Principles.



Following image shows screen to add target.

