Functional Layer Object Extender for Motive Dynamic Fulfillment

Priyanka D R¹ Dr. Mamatha G.S²
¹¹MTech Student ²Assistant Professor
¹²Department of ISE
¹²RVCE Bangalore, India

Abstract—Operations support systems (OSS), are computer systems used by telecommunications service providers to manage their networks. Different subdivisions of OSS are service fulfillment and service assurance. As part of this two products have been developed MDF(Motive Dynamic Fulfillment) and MDA(Motive Dynamic Assurance). In MDF the company is trying to ensure fulfillment of services is done completely. For example Inventory, provisioning of services. Hence the product is expected to interact with various telecommunication devices operating on different interface. It is for this reason that a robust software like FLOE(Functional Layer Object Extender) has been developed. FLOE is software network adapter that communicates with devices using interfaces like SOAP(Simple Object Access Protocol),TELNET etc. At the end it can be able to successfully communicate with various telecommunication equipments like switch, EMS (Element Management System),NE (Network Element),Router using different Interfaces like SOAP etc. Output from this equipments and are successfully able to convert it to the format(XML) expected by the remaining modules of the product. FLOE reduces effort in developing a new interface by approximately 70%. Because creating a new interface would take a lot of time and involves writing code, rules file and testing to make sure this interface can be used where an FLOE is already a software network adapter which has some predefined interface on top of which it can be extend some java class using some new rules file by which will be able to support a new interface.

Key words: Operations Support Systems (OSS), Motive Dynamic Fulfillment(MDF), Motive Dynamic Assurance(MDA), Functional Layer Object Extender(FLOE), Simple Object Access Protocol(SOAP), Element Management System(EMS), Network Element(NE)

I. INTRODUCTION

Operations Support Systems (OSS) are computer systems used by telecommunications service providers to manage their networks (e.g., telephone networks). They support management functions such as network inventory, service provisioning, network configuration and fault management. Different subdivisions of OSS have been proposed are service fulfillment and service assurance. Fulfillment of telecommunications services involves a series of supply chain activities responsible for assembling and making services available to subscribers. MDF(Motive Dynamic Fulfillment) product is to develop an extensible fulfillment solution[1]. Increase data accuracy of Inventory systems by discovering and reconciling what has been deployed. Enable an OSS platform ready for NFV SDN transformation [2]. In MDF Functional Layer Object Extender (FLOE) is used as an interface to communicate with the various Element Management System and Network Element to send request and obtain response which will later be modified and inserted into the DataBase. Functional Layer Object Extender (FLOE) is an application framework that is used as a configurable network adapter. A Software Development Kit (SDK) is provided to create, modify, or test FLOEs. The FLOE is a domain-specific application framework that is used in the distributed software integration domain. It allows users or developers to easily produce custom FLOE instances that integrate distributed software components to build various powerful distributed applications.

The main functionalities of FLOE are defining:

- Configurable activation rules, for each NE type, NE version, and service type. The configurable activation rules allows to define activation logic for creation, modification, or deletion of services with required parameters.
- Configurable rollback behavior. Rollback logic for activations that has failed midway before successful completion can be defined. Rollback call can be defined for each NE call.
- Configurable discovery rules for each NE type and NE version. The discovery rules allows to define what to discover and which queries to raise for a NE or NMS/EMS[3][4]. Each FLOE can support one or more functional domains, activation, discovery, test, and recovery.

II. DETAILED DESIGN OF FLOE

The FLOE (Functional Layered Object Extender) is an application framework that is used as a configurable network adaptor. A Software Development Kit (SDK) will be provided to create/modify and test FLOEs. FLOE SDK (Eclipse based plugin – will create the Rules files and other config files using a GUI – first release can be without the Eclipse plugin, but detailed docs on how to create both discovery and activate FLOEs)

A. Configurations Of FLOE

- Configurable activation rules, for each NE type, NE version, service type. The configurable activation rules enable to define activation logic for creation, modification or deletion of services with required parameters.
- Configurable rollback behaviour.
- It is possible to define rollback logic for activations that has failed midway before successful completion.
- For each NE call, its rollback call can be defined.
- Configurable discovery rules
- For each NE type, NE version, what to discover and which queries to be made to the NE or NMS/EMS[22].
- Each FLOE can support one or more functional domains, activation, discovery, test and recovery.

FLOE accepts requests from its north bound clients via incoming interfaces, invokes the FLOE core to translate the
requests into task trees according to the business logic contained in the rules files (XSLT based) and execute the task trees, and returns the task tree execution results to clients as shown in the fig 1. The FLOE core executes a task tree by interpretation using the Depth-First-Search (DFS) algorithm, starting from the root, and on reaching message nodes, sends messages to target systems via outgoing interfaces and handles the responses received as shown in the figure 1. During the execution, the outcomes of previous sub-tasks/messages may affect the execution of subsequent sub-tasks/messages.

Fig. 1: FLOE Architecture

a) Incoming Interfaces
The FLOE interacts with an external client system through an incoming interface to receive requests and send responses. To allow more client systems to use the FLOE to complete various tasks, FLOE supports several different types of incoming interfaces. To simplify the design and implementation of the FLOE core, the FLOE supports a uniform request or response structure. Requests or responses exchanged between the FLOE and its clients through different interfaces are converted to/from the uniform request or response structure.

b) Outgoing Interfaces
The FLOE interacts with an external server system or internal component to execute a message through an outgoing interface.

B. FLOE Core
The FLOE core provides the core functionality of the FLOE framework. The FLOE core maps the request received from the incoming interface to a list of tasks or messages, executes them by exchanging messages with external systems, and returns any result or exception to the client system as shown in the fig 2.

The major components of the FLOE core are:
- XSLT rule files
- Task builder
- TDL interpreter
- Message handlers
- Configuration
- Environment

The following figure illustrates the FLOE core:

Fig. 2: FLOE Core

a) XSLT Rule Files
XSLT is a transformation language that transforms an XML document into a document in another format, which can also be an XML document. An XSLT style sheet contains template rules (that is, XSLT instructions) that specify how a document is transformed into another document.

The FLOE uses XSLT to transform a request received from the incoming interface into a list of task or message definitions expressed in the Task Definition Language (TDL).

FLOE contains a set of XSLT rules files (that is, stylesheet files) that collectively map all the requests supported by it into task or message definitions. The XSLT rules files determine the capability of a FLOE, so each different set of XSLT rules files will result in a different FLOE.

b) Task Builder
The task builder is a factory object that creates task or message objects using the request received from the incoming interface. A request consists of a mandatory request name and three sets of optional name-value pair data, newParameters, oldParameters, and hints.

c) TDL Interpreter
The TDL interpreter interprets and executes task or message definitions expressed in Task Definition Language (TDL). TDL consists of two kinds of objects, task objects and message objects, both created by the task builder.

d) Message Handlers
Message handlers include message senders and response handlers. They work together to complete the handling of the messages contained in the message objects. Message handler is a generic name for either a message sender or a response handler.

III. METHODOLOGY
During start up, the FLOE initializes the environment so that it is ready to accept requests from the clients and execute tasks to fulfill the requests. After the environment is initialized, the FLOE enters the request processing loop to continuously process client requests as shown in the figure 3.
To process a request, the FLOE performs the following steps:

- Receives a request from an incoming interface.
- Converts the request data (parameters and hints) into a common data representation, called task data.
- Builds a task object with the current task data and task name.
- Interprets the task definition contained in the task object.
- After the execution of the task tree is finished successfully or unsuccessfully, converts any (maybe empty) results to the response format specific to the incoming interface from which the request was received and returns the response in that format.

IV. EXPERIMENTAL EVALUATION
During start up, the FLOE initializes the environment so that it is ready to accept requests from the clients and execute tasks to fulfill the requests.

A. Initialization
To initialize the environment, FLOE performs the following activities:

- Loads the system configuration file that contains parameters for the system, incoming interfaces, and outgoing interfaces. This step is mandatory.
- Loads and parses the XSLT rule files that contain the business logic and stores the XSLT templates in the template holder, which is a task name to the template map. This step is mandatory.
- Loads the response matching pattern files, which contain the default response matching patterns for outgoing interfaces, and stores the patterns in the response pattern holder, which is a file name to pattern map. This step is optional as it is not necessary to specify default response matching patterns for an outgoing interface.
- Initializes the session controllers with the parameters loaded from the system configuration file. Session controllers are responsible for creating and controlling CLI sessions. This step is optional as session controllers are only used in CLI outgoing interfaces.

B. Execute A Task
After the environment is initialized, the FLOE enters the request processing loop to continuously process client requests. Request processing is multi-threaded, that is, each client request is processed in a separate thread. However, the execution of the individual subtasks belonging to the same request is not multi-threaded, that is, subtasks of the same request are executed in the same request processing thread sequentially. FLOE uses this procedure to process a request. To process a request, the FLOE performs the following steps:

- Receives a request from an incoming interface.
- Converts the request data (parameters and hints) into a common data representation, called task data.
- Builds a task object with the current task data and task name. Initially the task name (of the root task) is the request name; later the task name of subtasks is the task name specified in the task definitions in TDL.

V. CONCLUSION
In FLOE Task Engine use to run continuously as a background process listening for events or requests to be sent to the south bound interface. If the process was terminated or killed, it would require human intervention to start process again and ensure that discovery does not fail. Moreover having the process run continuously in the background was not efficient either. The entire MDF product runs on JBoss application server hence it becomes necessary for FLOE also to run on JBoss so that it could successfully be integrated with the remaining product. This also solved the earlier problem of FLOE running as a background process because now FLOE is invoked by making a call to the EJB client, on which the FLOE is started once the request is sent and the response is obtained process is terminated.

REFERENCES
[2] Realizing the full benefits of network functions virtualization and software-defined networking - white paper
[6] Fatna Belqasmi, Jagdeep Singh, Suhib Younis Bani Melhem, and Roch H. Glitho Concordia,” SOAP-Based vs. RESTful Web Services A Case Study for Multimedia Conferencing” University Published by the
IEEE Computer Society 1089-7801/12/$31.00 © 2012 IEEE


