Mustafa Mamun Khondkar

# Business Support System Integration

Call Detail Record Processing and Order Management Implementation

Helsinki Metropolia University of Applied Sciences

Bachelor

Information technology

Thesis

12 September 2015

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages | Mustafa Mamun Khondkar<br>Business support system integration<br><br>44 pages |
| Degree | Bachelor of engineering |
| Degree Programme | Information Technology |
| Specialisation option | Software Engineering |
| Instructor(s) | Ari Aalto, Head of R&D, Tampere<br>Dr.TeroNurminen , Department head |

The objective of the project is to create a new BSS to solve the issues that telecommunication service providers are facing today and find the best ways for network operators to run their business. The project is carried out by following the agile development method. JIRA is used to track the issues and roadmaps. The majority of the project is built on new technologies which makes it scalable. Additionally the architecture of the project is built keeping configurability in mind. It will be able to adopt any change or upgrade in the operator business. Furthermore it will help the network operators to improve their bottom line by giving them a complete and constant overview of their business. Eventually it is about creating a better experience forend-users.

This paper describes the prototyping of a call detail record processing engine using Node.JS, implementation of a REST layer on a SOAP API,development of order management system and their outcomes.

The performance of the call detail record processing engine created by this project is quite satisfactory and the architecture is followed by different other modules of the project. The REST layer is running without any crush since its deployment. The order management is still an ongoing project. However what has been developed is stable and satisfactory.

| | |
|---|---|
| Keywords | BSS,Node.JS, REST, Swagger, SOAP, API, Apache, Avro, CDR, Order Management. |

## Contents

# 1 Introduction

Business Support Systems (BSS) are a set of software components and functions interconnected together to grant the monetization of the communication service providers (CSPs), or simply the operators. It also allows the operator to collect his/her money ontime and charge for services as per the contract agreement with the end users such as prepaid orpost-paid. BSSs usually tend to be customer-facing systems. These are used mainly to store customer-related information and are involved in transactions tha-trequire this kind of information. One example could be charging systems. Thiscon-tainthe prepaid customer profiles, the tariffs, the plans and so on. Network check the charging system to know if a particular subscriber has enough money in his account to do a specific action such as call or browse.

BSS is dealing with five main big areas which are customer relationship management, product management, order management,billing and revenue assurance. Traditionally BSS was limited in creating orders, customer support and billing. However the complexity and the scale of today's multivendor networks and service offerings are staggering. Network operators want to offer services such Spotify and Netflix to their customers. As a result an enormous amount of data needs to be processed, analysed and translated into ways to improve the customer experience, with 50 billion devices expected around the world by 2020 it will not get any easier. Furthermore customers calling for help do not want to wait for an answer why something is not working or how much data they are allowed to download or to find out about their invoices. Operators need to have complete, end-to-end overview and control of the information in real time, they need to have an answer ready before the customer even calls customer support.

The goal of the project is to create a new BSS to solve the above issues and find the best ways for network operators to run their businesses. The majority of the project is built on newtechnologies,which makes it scalable. Additionally the architecture of the project is built keeping configurability in mind. It will be able to adopt any change or upgrade the in operator business. Furthermore it will help the network operators to improve their bottom line by giving them a complete and constant overview of their business. Eventually it is about creating a better experience for end-users.

# 2 Theoretical Background

## 2.1 History

Before 1980the task of the telephone company such as maintaining network inventory, takingorders, configuring network components, managing faults and collecting payments, servicesprovisioningfor example, testing and line assignment, were carried out manually. After realizing that many of these activities could be computerized, a number of computer systems and software applications evolved to automate these activities in the next few years. Some of the old applicationsare Trunks Integrated Record Keeping Systems (TIRKSs), Remote Memory Administration Systems (RMASs), Service Evaluation Systems (SESS) and so on. That is how the phrase Operations Support Systems (OSS) came in practice. OSS is a set of applications that help theCSPs manage, monitor, analyze and control a telephone or computer network.[19.]

Compared to OSS,BSS is a newer term. It typically indicates the business system that directly deals with the end customer. The typical operation of BSS is taking orders, supporting customer care service, processing bills, collecting bills, invoicing, billing inquiries and trouble ticketing if necessary. BSS and OSS systems are often called together B/OSS. In the past the OSS system itself includedboth the business system and the networks. Even currently some of the network specialists, service providers and system integrators use the term OSS to include both the business system and network.[19.]

## 2.2 BSS Core Areas

BSS systems, as the name implies, are generally geared towards the operation of the business, such as order entry, order fulfillment and customer relationship management. BSS also encompass back-office activities such as service assurance and trouble ticketing of OSS which are initiated directly by contact with the customer. The five main roles of BSS are discussed below.

### 2.2.1 Customer RelationshipManagement (CRM)

CRM system processes are very important to CSPs to deliver consistently superior customer experience. They also need to come up with a unified product catalog for the customer regardless of geography and market segment, and maintain a unified master database. Furthermore, they need to figure out the product thathas most growth potential and also identify the high-value customer. A perfect CRM solution should have the capability to coordinate, integrate, and organize customer communication across mediums. It should provide a holistic, 360-degree view of customers across the globe, and multilingual facilities. It should be a cloud-based solution which includes business applications that will intensify customer management, which directly leads to profitability. Additionally it should provide CSPs with the capability to maintain the entire lifecycle of a customer effectively and improve the relationship with the individual customer as well as the large enterprise customer. Finally it should be easy and quick to implement, and should help CSPs to minimize the incident of unpleasant customer experiences due to factors such as network issues and provisioning. Figure 1 shows how a CRM works in BSS. [2.]

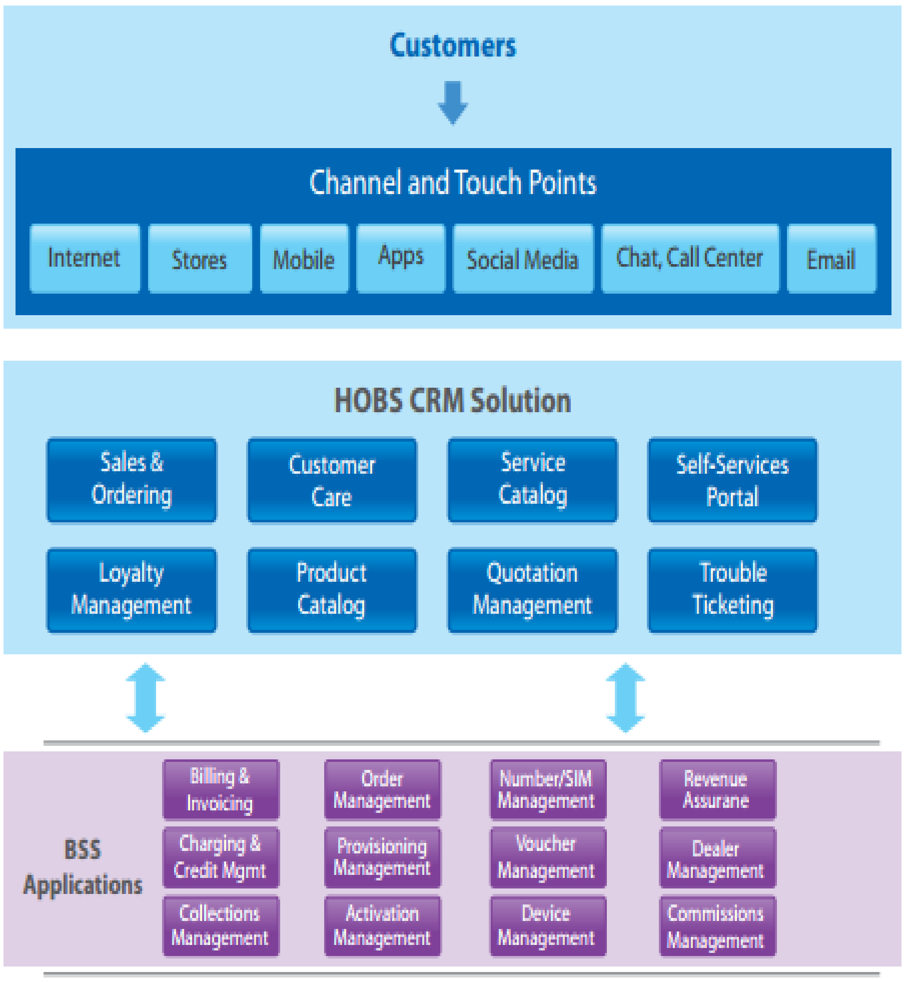


Figure 1: A typical CRM in a BSS. Reprinted from TATA [2].

Several functional areas are covered by the typical OSS/BSS CRM solution:

- Sales and Ordering finds and introduces new customers, enables faston boarding, supported by advance validation which minimize order errors and eventually accelerate the time to revenue.[2.]
- Hierarchy Management **e**nables sharing complex charge/discount across multiple member assignedresidential and business segments, inside or outside the hierarchy.[2.]
- Unified Product Catalog reduces inconsistencies in product information across systems and launch products faster.[2.]
- Customer Care and Adherence obtain a 360-degree view of customers, which capacitates a new level of service apart from what commercial metrics provides. End-to-end care leveraging pre-integrated operations helps to meet key performanceindicators (KPIs) and service levelagreements (SLAs). A loyalty engine with flexibility and pleasant service helps to achieve loyalty and rewards.[2.]
- Self-care portals enables the customer to manage most of the things by themselves hence customer care agentsare capable of adding more values to the business. [2.]
- Authentication and security is maintained through captive portals.[2.]

### 2.2.2 Product Management

Product management is the process of conceiving, planning, implementing, testing, distributing, delivering, launching and withdrawing products in the market. The purpose of product management is to create customer value and measurable business benefits. It identifies customer problems to create new products that outwit the competition over a sustainable period of time. A good product management system should help the business owner to grow product performance in the market, monitor product performance, and report on the results, responds to the issues and plan for the future. Furthermore it should enable the business owner with the capability to change the product to adopt the changing market conditions. [13.]

Competition is intense betweenCSPs, and new attractive products are the key to attract new customers as well as maintain old ones. For many CSPs, who are struggling with old legacy OSS environments and traditional product technology silos, this is not good news. For them, new product introductions mean multiple system updates, complex integration, extended development times, and high costs. It even takes far too long for them to launch a new product that are similar to existing ones. Meanwhile, customers always want everything immediately. They want a product to be available, they want to buy the product online and they want to buy it at a cheap price. In the current approach the amount of automation is possible with limited resources, which is a difficult proposition. Hence to reduce time and cost CSPs obviously need a unified product management process, and greater process automation to enable online ordering. Hence good product management should enable both. It should change the new product model from rebuild to reuse. Additionally it should be a solution that automates, consolidates, and changes a process from pre-order to order fulfillment. [1.]

The product management system should provide the CSPs with a master database to manage all their product. The database should impose all the rules and logical bindings among the products. Every product should work as a single entity so that CSPs can create bundle or offers by adding the products together. It enhances the reusability of the product. The following is a typical product management architecture in BSS.

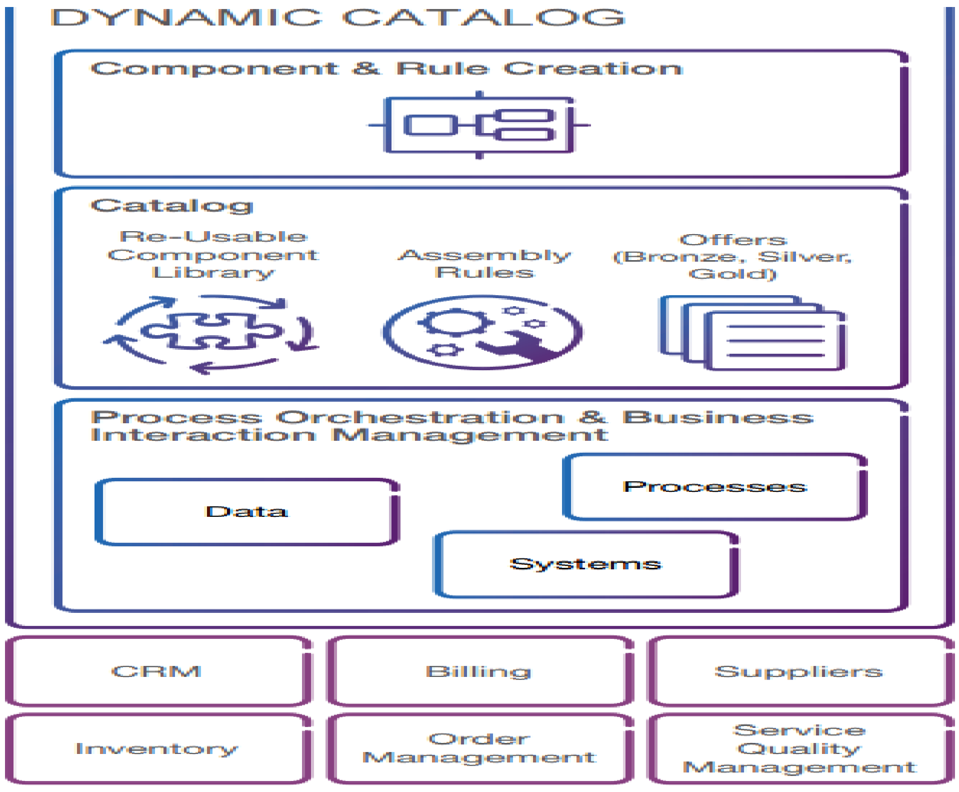

DYNAMIC CATALOG
Component & Rule Creation
Catalog
Re-Usable Component Library
Assembly Rules
Offers (Bronze, Silver, Gold)
Process Orchestration & Business Interaction Management
Data
Processes
Systems
CRM
Billing
Suppliers
Inventory
Order Management
Service Quality Management

Figure 2: Product management in BSS. Reprinted from Ericsson [1].

Only an intelligent and skilled product management system can gain success in the field of software-intensive products. To accomplish a win in the telecom business the product management system should know what functionalities and aspects the product should offer to which group of customers and at what time.

### 2.2.3 Order Management

The Order management system is a very important and complex module in the BSS application stack. It helps the service representatives and customers to create a new subscription, modify an existing subscription, suspend or revoke a subscription, terminate or activate an account, and terminatea subscription. A picture of a typical order management system is given below.

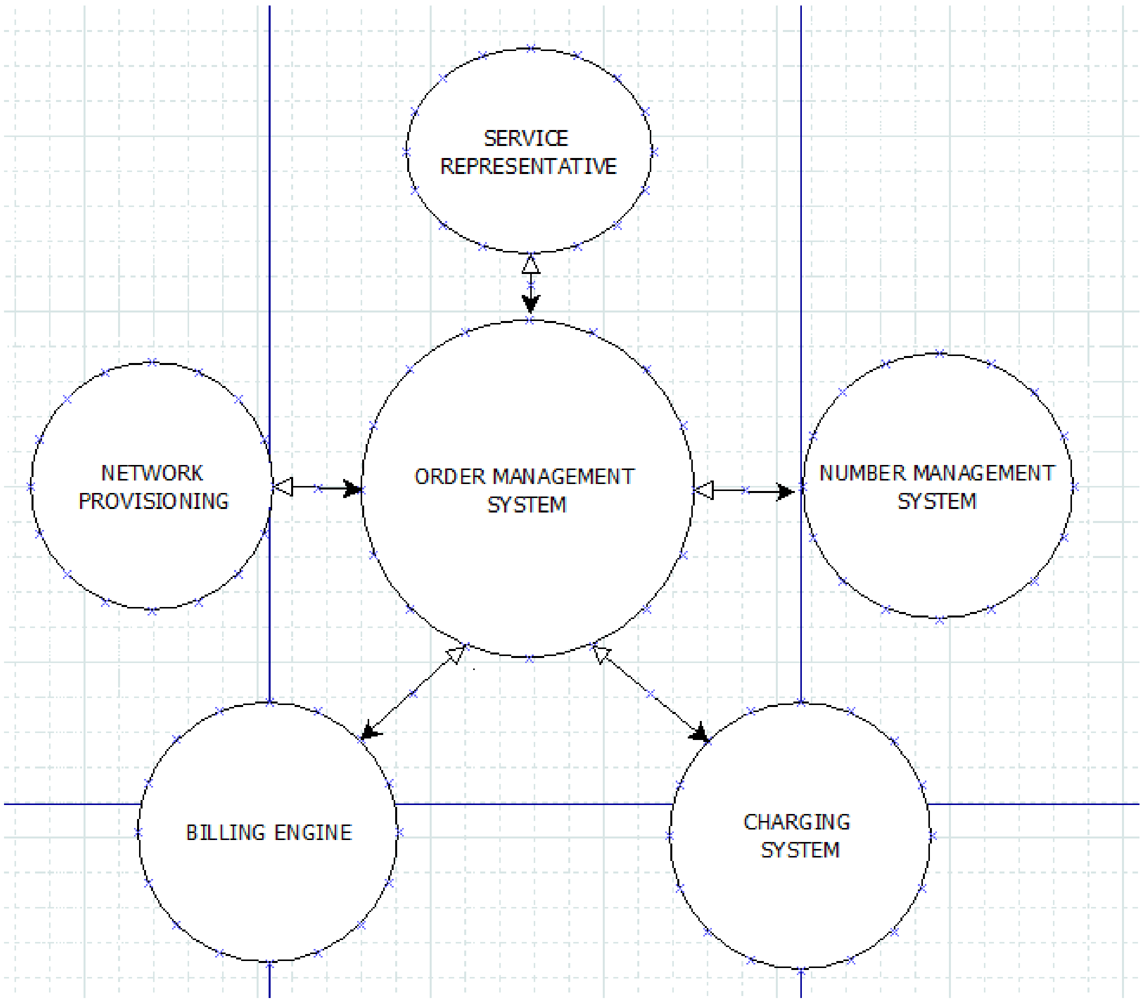


Figure 3: Architecture of a typical order management system

The typical operations of an order management system are thefollowing

**The create subscription process** takes orders from the customer. The order typically includes, for example customer personal details, prepaid or post-paid account detail, charging contracts, offerings, plans and allowance. After taking the order the order management system validates the order. If the order includes any faulty data, the system will immediately send a response about the fault. If the order pass validation the system proceed to next operation. At the next step the system provisions the network for the customer and then does integration with the billing system if the customer orders a post-paid subscription. In the billing integration process the system creates a billing account, adds an offering to the billing account, creates a service account and activates the service account.

Having done the billing integration the system will create an account for the subscriber in the charging system database with the customer provided data. Information about the subscriber account, prepaid or post-paid account, allowance, contracts andplans are normally stored in the charging system. After creating an account in the charging system the order management system will finally reserve the MSISDN number assigned to the user in the number management system and finish its work. The order management work like a waterfall process and if any of the integration process fails, it will raise a trouble ticket for human interaction.

**The modify subscriptionprocess** is used to modify customer information in the billing system and in the charging system. A typical modify process normally does not include the network provisioning and MSISDN reservation.  In the modify process the system takes an order from the user and validates it. After validation the system directly goes to the billing system and charging system subsequently and changes the customer information there.

**Suspend or revoke subscription** means pausing or resuming recurring charges in the billing system and blocking or activating a prepaid account in the charging system. This operation also does not include the network provisioning and MSISDN reservation. In the case of a suspend operation the order management system takes the order

and validates it. If validation passes, it will pause or resume all the recurring charges in the billing system and block or unblock the prepaid account in the charging system.
A subscriber can have more than one account and he/she may want to terminate or activate one of his/her accounts. The **Terminate or activate an account process**is to solve this issue. In this process the system takes data from the customer and validates it. If the validation passes, the system will subsequently do network provisioning, termination and activation of a service account in the billing system, closing and activating a prepaid account in the charging system and MSISDN release and reservation.

**The terminate subscriptionprocess** is used to terminate the subscription of a customer from the system. In this process the system takes an order from the customer and validates it. If the validation passes,it will proceed to do network provisioning and closeall the network provisioning for this specific subscription. After that it will terminate all the service accountsin the billing system and close all the prepaid accounts and charging contracts in the charging system. Finally it will release all the numbers associated with the subscription from the number management system.

### 2.2.4 Billing

The billing system converts the service into money and returns it back to the service providers. CSPs setup networks and manage them to allow the customer to communicate with each other and in return charge bills from the customers. In the competitive market of telecommunication billing has become an important strategic tool for the CSPs. For CSPs bills are the most important element to maintain their revenue and for the customer bills are the yardstick to evaluate the service expectations.

Before 1990 billing was an in-house process. Circuit switched telephone calls were the main focus area in those days. The parameters used to charge was distance and duration. However with the advancement in the telecommunication sector triggered a need for a more complex billing system. Today the main priority of the CSPs is to create basic revenues and profits. Furthermore producing accurate bills is very important to make revenue as well as to serve the customer effectively.

The billing system collects the call detailrecords (CDRs), rates them and finally calculates them for telecommunication service providers. The basic architecture of a billing system is given below.

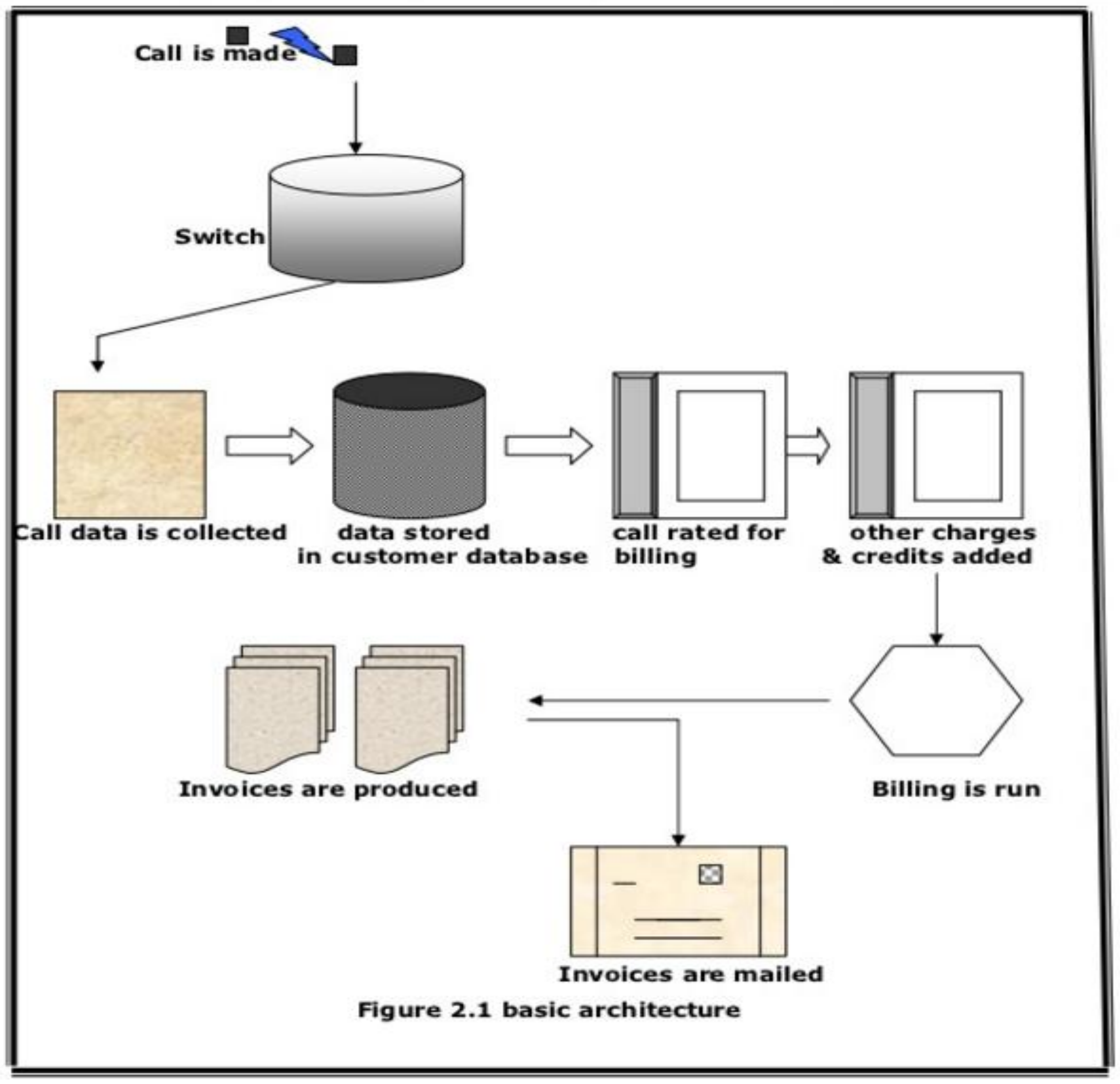


Figure 4: Architecture of a typical telecommunication billing system. Reprinted from Sharker [14].

When a call is made or any network activities like browsing the Internet or sending a text or voice message is done, a record about the event is stored in the network switch. This record is called call detail record (CDR). A CDR includes detailed information about the event and the customer who created the event. After creation the guiding engine creates charges for the event. It checks the customer database, finds the plans the customer has and charge the event accordingly. After guiding has been done the CDR gets rated. This process gives the event a value to be charged at the time of billing. Finally it is saved in a file system in the cloud server.

Billing is done once or twice a month. The billing engine collects all the rated CDRs stored for the last 15 or 30 days and calculates the charge. After that it adds promotions or discounts with the charge if necessary. In addition taxes are also applied. When the billing is complete, it createsan invoice and sends it to the customer address. The address can be a postal of email address.

### 2.2.5 Revenue Assurance

Revenue assurance is a process of measuring the achieved revenue against forecast, accounting for any discrepancies, verifying the amount being billed, protecting and optimising the revenues and profits. In an ideal world, revenue assurance encompasses every step in the revenue process all the way from the transaction to the accounting ledger. The revenue assurance process is integrated within the overall enterprise risk management of the company. It covers all the revenue-related risks ranging from revenue leakage through the revenue recognition in the financial statement. Additionally it manages people, processes, and technology in an integrated way to ensure the maximum revenue and minimum costs.

Revenue assurance is done by using the following methods

- Identifying the key attributes of every service that is in operation and can affect the revenue.
- Identifying the affected data sources associated with the key attributes such as switches, mediation components, roaming charges, and cross operator calls.
- Mapping the data source to the information model
- Identifying the data flows across the data sources such as identifying the location of the data source and to where the data is being sent from each of the source
- Enabling controls to ensure the integrity of data related to revenue across all data flows
- Defining thresholds for loss at each point. This should be derived from the target threshold on an end to end reconciliation, thresholds such as rejection tolerance and time to receive CDRs.
- Cross checking the devise mechanism from the first point to the last point in the data flow.

Revenue leakage across the revenue chain remains a challenge for the operators. Only 2.5 to 37.5 percent of total leaked revenue is recovered in most cases. This indicates that majority of the leaked revenue remain uncovered. Although the percentage of leaked revenue is one to three percent globally. However the total amount of revenue gets leaked every year even in this rate is huge. [15, 11.]

# 3 Methods and Materials

## 3.1 Node.JS

I have used Node.JS to develop my entire project. The reason behind choosing Node.JS is its scalability, ease of use, non-blocking I/O and performance. Furthermore we are using REST API and JSON data structure to communicate between application components. It is possible to create a very elegant REST interface using Node.JS and Swagger easily.Node.JS allows the developer to build a scalable network application using JavaScript on the server-side. Underneath the covers of Node there is V8 JavaScript Runtime. This is the same runtime used by Google Chrome. Node provides a wrapper on it and adds more functionalities to create a network application.Node.js was developed by Ryan Dahl in 2009 and its latest version is v5.0.0. The definition of Node.js as put by its official documentation is as follows:

> "Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices" [4.]

Node.JS is an open source runtime environment for developing a server side application. Node.JS applications are developed using the JavaScript programming language, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.Node.js also provides a rich library of various JavaScript modules through the node package manager (NPM) which eases the development of the web application using Node.JS to a great extent. NPM is a repository for node packages.  It is an open source library. Anyone can upload or download a package from NPM. [4.] To use any package the user just needs to include the package in his/her application with the following command:

```
npm install <package-name>
```

After that he/she can require the package in the application and use it.

The following are a few of the important features which are making Node.JS the first choice of software architects.

- All operation of Node.JS is asynchronous, which means it does not wait for an operation to return data. The server moves to execute the next line of code. When an operation is done, it calls its callback function with an event and does its operation with the return data. [4.]

- Google Chrome's V8 JavaScript Engine is built using the C programming language. Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution. [4.]

- Node.js uses a single threaded model with event looping. It continuously checks for events. When a request comes in, it triggers an event and this event calls the callback function written for this event. Events are processed within an event loop one at a time. [4.]

- Node.js applications simply output the data in chunks.These applications never buffer any data.[4.]

- Node.js is released under the MIT license[4].

## 3.2 Apache Kafka

Kafka is a distributed, partitioned, replicated commit log service designed for processing of real time activity stream data such as logs and metrics collections. Kafka maintains feeds of messages in categories called topics. The process which publish messages to the Kafka is called a producer and the process which consumes the messages from Kafka is called a consumer. Kafka is run as a cluster comprised of one or more servers each of which is called a broker. Simple, high performance and language onistic TCP protocol is used to connect the clients and the server.The following picture is a high-level view of how Kafka works. [3.]

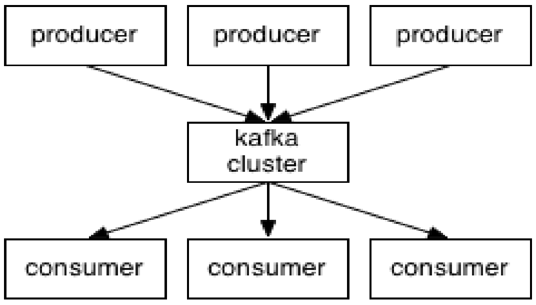


Figure 5: Architecture of Kafka. Reprinted from Apache[3].

Kafka can be used for multiple purposes. Some important uses of Kafka are given below.

- Kafka is a good replacement of the traditional messaging system. Messaging systems are used to decouple processing from the data producer and to buffer unprocessed messages. In comparison to other messaging systemsKafka has better throughput and fault tolerance. Additionally it provides built-in partitioning and replication which makes it perfect for large scale data processing.[3.]

- Kafka was built under the Apache foundation for tracking user activity in LinkedIn. User activity means page views, searches and any other action the user might take. These activities are published to Kafka as event per topic. These events are useful for a range of applications like real time monitoring and processing or loading into Hadoop for offline data processing, reporting and warehousing. [3.]

- Kafka can be used to monitor operation in data pipeline and aggregate statistics from a distributed application to produce centralized feeds.[3.]

- Log aggregation means saving the log file in a central repository or server for further processing. Kafka makes log aggregation easier by abstracting away the details of the file. With Kafka, log data can be saved as a stream of messages which enables to lower latency and support for multiple data sources. In comparison with other log aggregation systems Kafka provides betterperformance, fault tolerance due to replication and low end to end latency.[3.]

We have used Kafka extensively in our projects. In the CDR processing project we have used it as a message queue between the CDR processor and the consumers. Alongside, in the development of Order Management I have used Node micro service framework. Kafka is used there to bind the micro services together.

## 3.3 Serialization with Apache Avro

Apache Avro is a data serialization framework developed by the Apache foundation.LinkedIn is using Avro internally for message encoding with Kafka. Every message passed on the wire has an accompanying schema. Written in JSON, the schema enables a number of features including message compaction, documentation, validity checking, versioning and discovery. In the producer the JSON data is serialized into Avro format which produces a binary, compact representation of data. Avro's means of facilitating schema evolution is more elegant and sophisticated than the other contenders. A typical Avro schema looks like the following:

```
{
"type": "record",
"name": "Student record",
  "fields" : [
    {"name": "name", "type": "string"},
    {"name": "address", "type": "string"},
    {"name":"id", "type":"int"}
  ]
}
```

Avro schema, unlike Google Protobuf or Thrift, are consulted at run time and not necessarily backed in at compile time.  However to use them a high availability schema repository is required. [6.]

When placing Avro encoded message onto a queue, the developer needs to stipulate what schema to use when decoding the message. There are multiple ways to accomplish this. The schema can be included directly but this obviously negates the message compression advantage.Another alternative is to encode a batch of messages of the same type together and specify a single schema that applies to all of them which is not very useful for practical use. As last option a reference to a schema can be passed which is held in a repository. This last option is used by LinkedIn. They deployed their own HA schema repository and carried out proprietary modifications to the magic bytes in the message header to inform the clients about the encoding scheme and schema identifier so that the consumer can retrieve the specific schema from the repository. Confluent offers an open source schema repository implementation for Kafka that relies on Kafka itself to store schemas in a special topic with additional index/offset cache to quickly locate a given schema. It integrates in well with Kafka but there is a serious downside to using it. It requires developers to take the Confluent version of Kafka and route all the non-Java producers and consumers through a REST web service provided by Confluent.

We already decided to use Kafka as the intermediarytool between CDR processor and billing system. Apache Avro encoded data has a great performance inside Kafka message queue and therefore we decided to use Apache Avro as our serialization framework. Serializing data with Avro gave us two great features. It encodes the data which cannot be decoded without the specific schema and great performance boost.

## 3.4 Sublime Text

**Sublime Text** is a sophisticated and excellent text editor designed for the programmer who likes to shuffle code around. It is actually a cross between a text editor and an IDE. It allow the developer to concentrate on more important issues by automating boring and repetitive tasks. It works on OS X, Windows and Linux. There are many useful open source plug-ins that can be added to it.Plug-in like **JSHint** and **JSLint** are very usefulto write clean and error-free JavaScript code. Some of the excellent feature of Sublime is given below. [12.]

- **Sublime Text** provides a recursive search and replace feature. Hence developers do not need to go through each line of the code. This is a very important features at the time of aggressive refactoring. Compared to many other text editors the search and replace feature in Sublime works better. [12.]

- Opening a project and closing is very fast in Sublime. Compared to Eclipse it takes about 10 seconds to open and close a project where Eclipse takes two minute only to open the project. Opening a file, searching a file, switching between files or closing a file is amazingly fast. [12.]

- Almost everything in Sublime is customizable with a simple JSON file. Features such as key bindings. macros, snippets, compilation, menus and so on can be customized with the JSON file. [12.]

- The "go to anything" feature allows developers to search a file with limited key-strokes or instantly jumpto symbols, lines, or specific word. This feature gets triggered by typing **Ctr+P** .[12.]

- **Sublime Text** offers a number of plug-ins to make life easier for programmers. It works like App store. Sublime text has a package manager repository from where programmers can search, download and install a required plug-in with minimal effort. [12.]

## 3.4 Red Hat and CentOS

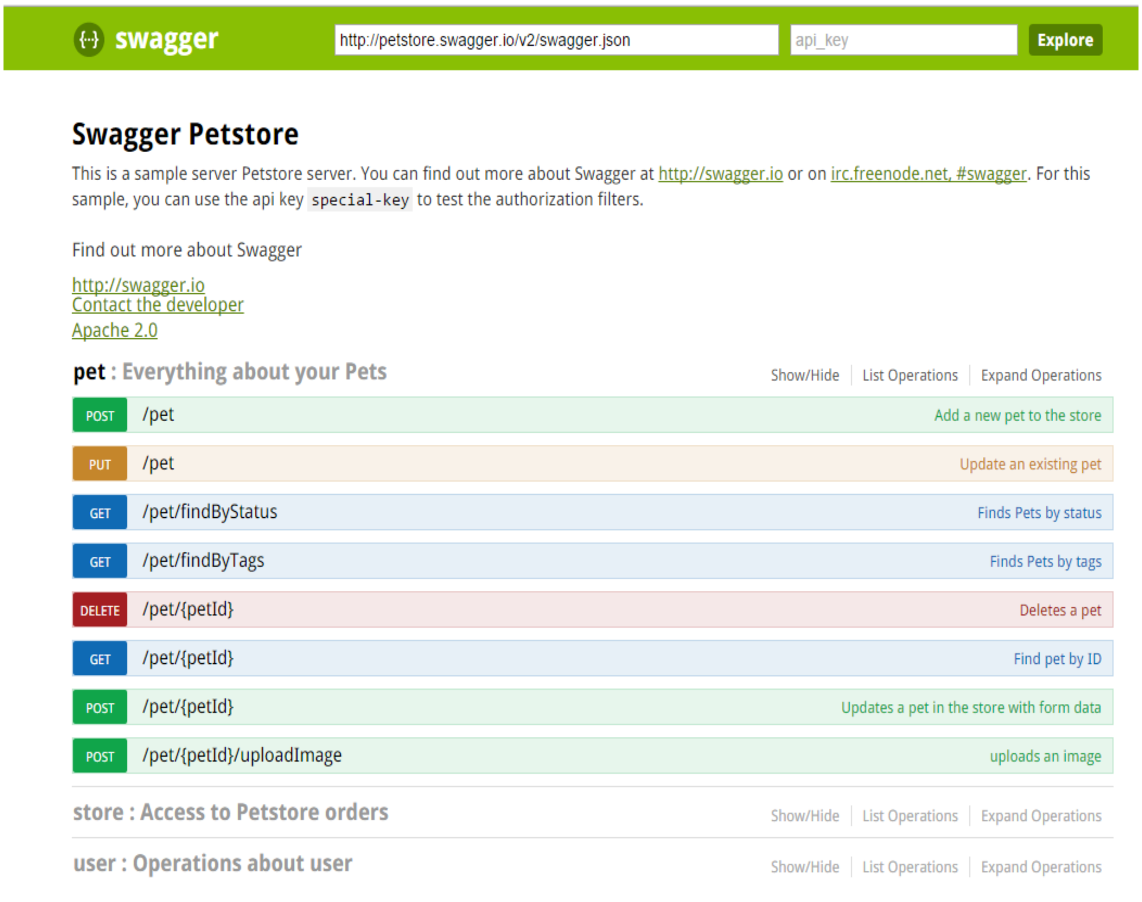
Red Hat Enterprise Linux operating system is the leading open source platform used in the industry for server maintainers. It is supported by multiple system architecture such as Intel (x86_64), AMD, IBM POWER and IBM System z. Software giants such as Google or Facebook are using Red Hat to maintain their servers. It is provided on a per-physical-system annual subscription basis. It does not have any hidden costs. Additionally it supports customers for unlimited incidents and allows them to upgrade for free.Red Hat offers encryption, memory page sharing, ballooning,live migration, load balancing, snapshots, flexible storing, rapid provisioning, desktop pooling, search base management, auto suspension and so onfor the customers. [7.]

The company I have been working with are also using the Red Hat cluster to run the application. After developing the application locally I had to install the application in the Red Hat server for further testing and release.

CentOS stands for community enterprise operating system is a community supported open source operating system based on Red Had Linux.The CentOS developer are trying to deliver a free and enterprise-class platform while keeping it computable with Red Hat. They are using the open source Red Hat code to create it. CentOS is a completely free operating system. The community provides technical support by mail, web forum or chat rooms. The project is not associated with Red Hat and gets no support from them. It is running with the donation from the individual user or company sponsors. I have used CentOS in my local computer for application development. The reasons behind using CentOS is its compatibility with Red Hat. [8.]

## 3.5 API documentation with Swagger

Swagger is a joint open-source project by several vendors aimed at providing representational language for RESTful service oriented end points. It is used by hundreds of companies and is supported by many vendors such as Apigee, Mulesoft and IBM.It has many open source sub-supporting projects such as Swagger UI to create aninteractive web UI for the end point, Swagger editor to write Mark Down and Swagger SDK to build API in multiple languages. Swagger also ships with schema validation and security validation. In Swagger developers can specify whether a parameter is Array, Object, String or Integer type. Swagger will response to the user with an error if the wrong type of value is given to a parameter. Furthermore a parameter can be declared as mandatory or optional in Swagger. [5.] Web UI of a swagger API end point looks like the following screenshot.



Swagger Petstore

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Find out more about Swagger

http://swagger.io
Contact the developer
Apache 2.0

pet : Everything about your Pets Show/Hide | List Operations | Expand Operations

| | | |
|---|---|---|
| POST | /pet | Add a new pet to the store |
| PUT | /pet | Update an existing pet |
| GET | /pet/findByStatus | Finds Pets by status |
| GET | /pet/findByTags | Finds Pets by tags |
| DELETE | /pet/{petId} | Deletes a pet |
| GET | /pet/{petId} | Find pet by ID |
| POST | /pet/{petId} | Updates a pet in the store with form data |
| POST | /pet/{petId}/uploadImage | uploads an image |

store : Access to Petstore orders Show/Hide | List Operations | Expand Operations

user : Operations about user Show/Hide | List Operations | Expand Operations

Figure 6: Swagger UI.

Swagger provide an API key field in the UI. It is possible to send the API key value as an authorization header and perform authorization afterwards.For Node.JS development Swagger can be installed through **Node Package Manager.** To install it globally the following command is required to type in the Windows command prompt or Linux shell.

```
npm install swagger –g
```

After installing the Swagger the following command will initiate the creation of a new swagger project.

```
swagger project create
```

After inserting the above command it will ask for the project name followed by the framework to use for this project. When the inputs are given it creates a new Swagger project with the name and the framework input.

## 3.6 GitHub

GitHub is a version controlling system based on Git. Git is auniquely designed, high-speed open-source distributed version control system. Linus Torvalds initially developed Git for Linux Kernel development.  Every Git working directory works as a complete repository and is capable of tracking all the changes made even though it is not dependent on network access or the central server.  Git is developer’s best insurance against accidental mistakes or a system crush. Git enables its users to commit changes to remote branches, revert the changes back, compare the changes over time, see who modified what, what is modified in which commit, control modifications by collaborators with the permission of admin/owners, tag specific point and many more.[11.]

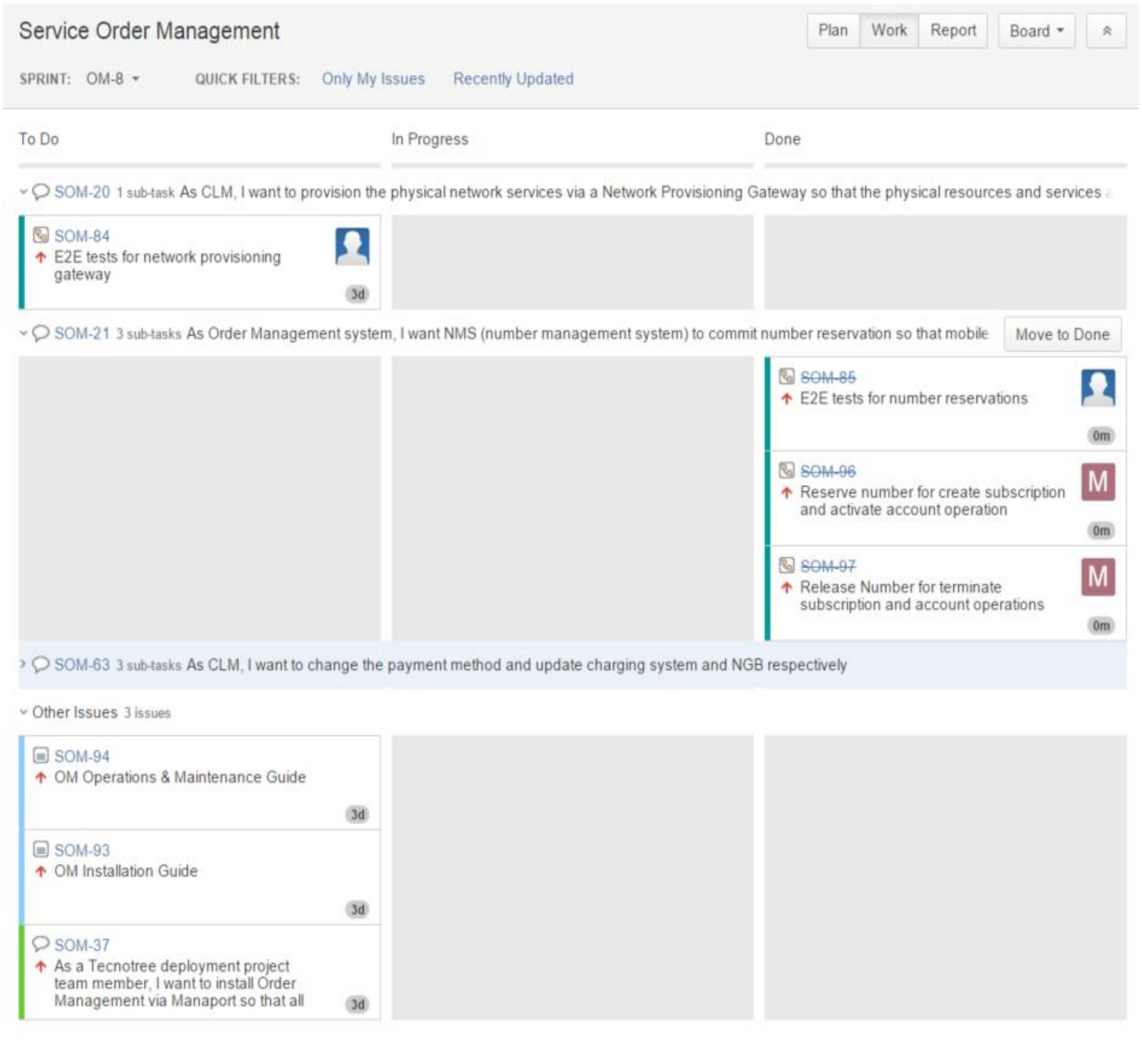
While Git is a command line tool, GitHub provides a web-based graphical interface that works on top of Git. It can also be treated as a social platform to share knowledge and work. It also provides access control and several collaboration features, such as wikis to document the project architecture or store some other documents and basic task management tools.

We are using GitHub extensively in our project. We do our release through the master branch. Under the master branch we havea development branch for testing. For solving issues or items we create our own branch and do the work there after completing the work we merge back to development and do the testing. When the testing is done, we merge back to master and do the release at the end of every month.

## 3.7 AtlassianJIRA

JIRA is a widely used issue tracking and project management tool. It enables the companies to prioritize, assign, track, report and audit issues from software bugs and help-desk tickets to project tasks and change requests. It can be used more than just as an issue tracker. JIRA is an extensible platform that can be customized to manage business process. It improves productivity by cutting down the time wasted on tracking issues and coordination. Furthermore it improves quality by ensuring all the tasks are recorded down with all the details and followed up till completion. [9.]

For issue tracking JIRA offers a task board view. There are three columns in the task board view,**To do**column, **In progress** column and **Done** column. When a task is created, it is saved into "To do" column. Every task has a description, type and user stories. What is the task all about and the acceptance criteria for the task is written in the description field. The type specifies what type of task it is. Is it a new task or improvement of an old task or a bug fix? The story point defines the number of working days required for one developer to complete the task. Normally one story point is equal to one working day of a worker. When a developer starts working on a task, he/she moves the task in**In progress**column and after completing the task by following certain criteria specified in the description of the task the developer resolve the task and move it **inDone** column. Figure 7 shows JIRA task board view of our Order Management project.



SOM-20 1 sub-task As CLM, I want to provision the physical network services via a Network Provisioning Gateway so that the physical resources and services

SOM-84 E2E tests for network provisioning gateway 3d

SOM-21 3 sub-tasks As Order Management system, I want NMS (number management system) to commit number reservation so that mobile

Move to Done

~~SOM-85~~ E2E tests for number reservations 0m

~~SOM-96~~ Reserve number for create subscription and activate account operation 0m

~~SOM-97~~ Release Number for terminate subscription and account operations 0m

SOM-63 3 sub-tasks As CLM, I want to change the payment method and update charging system and NGB respectively

Other Issues 3 issues

SOM-94 OM Operations & Maintenance Guide 3d

SOM-93 OM Installation Guide 3d

SOM-37 As a Tecnotree deployment project team member, I want to install Order Management via Manaport so that all 3d

Figure 7: JIRA task board view.

JIRA can also be used for reporting. It delivers real-time, relevant information in a convenient format. Additionally it enables the management to have clear visibility of the situation. Furthermore JIRA can be used as a roadmaps as it enables the management to know what is outstanding and when issues are scheduled to finish.

## 3.8 Continuous Integration with Jenkins

Continuous integration is a development practice that requires the developers to integrate continuously into a shared repository several times a day. Each commit is then verified by an automated build, allowing the teams to detect any bug right away. It saves time by replacing the more traditional testing which used to be done after the whole process was completed. About continuous integration Martin Fowler said in his blog:

> "Continuous Integration is a software development practice where members of a team integrates their work frequently, usually each person integrates at least daily which leads to a multiple integration per day. Each integration is verified by an automated build to detect integration errors as quickly as possible"[18.]

Jenkins is an award-winning open-source continuous integration tool written in Java. It monitors the execution of a repeated job such as building a software project or jobs done by cron. It is a server based system running in a servlet container such as Apache Tomcat. Jenkins provides an easy-to-use continuous integration system making it easier for the developer to integrate changes into the project and making it easier for user to obtain fresh build. The automated continuous build increase productivity. [16.]We are using Jenkins for continuous integration and testing of our project. Our GitHub development branch is connected to Jenkins. Jenkins is running our test code and showing the result through its dashboard.

### 3.9 REST

REST stands for Representational State Transfer. It is not a standard nor a framework, rather an architectural style for developing web-based systems. REST was first proposed by Roy Fielding in 2000. The goal behind the REST architecture is to utilize the basic characteristics of the web, which made the web successful. It utilizes the basic HTTP verbs to accomplish the basic CRUD operations: GET for reading or fetching data, POST for sending data, PUT for updating data and DELETE for deleting data. REST defines six architectural constraints that a system architecture should comply with to obtain scalability. [10,129.]

- **Client server paradigm** improves portability of UI by separating the front end concerns from the back-end concerns. Furthermore it improves the scalability by simplifying the server component and finally allow the component to evolve independently. [10,130.]

- Theclient request should be independent and completely understandable by the server. All info needed should in encapsulated in the request. Hence the session state is kept in the client side. It improves visibility, reliability and scalability.[10,130.]

- Interaction are partially or completely eliminated depending on the label on the data whether it is **cacheable** or **non-cacheable**. This improves network performance.[10,130.]

- The main feature of REST is to provide a **uniform interface** between the components of a distributed application. In uniform interface the identification and the manipulation of resources are done by following some standard procedure. It simplify architecture and encourage evolvability. [10,130.]

- Layers are created to hide the complexity or to provide more standard interface. It can be used to encapsulate the legacy service or protect the new service form legacy client.[10,130.]

- This is optional in REST architecture. It means dynamic execution of code in the frontend.[10,130.]

## 3.10 Agile development

Agile software development refers to a group of software development methodologies that are based on similar principals. Agile methodologies generally promote a project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages team work and accountability, a set of engineering best practice that allows for rapid delivery of high-quality software and a business approach that aligns development with customer needs and the company goal. Agile methodologies give priority to individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan.[17,160.]

We are using the Scrum agile software development methodology in our office. In each sprint we solve a big task. This task is called **Epic.** Each Epic is divided into several **User Stories**. The Scrum master usually creates all these user stories and adds description to them. In each sprint planning meeting the Scrum master asks for a **Story Point** from all the developers for each **User Story** and decide the **Story Point** for each **Users Story** on basis of the developers answer. The work needed to complete a user story is defined by the **Story Point**. The more story point a user story has the more work is needed to complete the user story.

After deciding about the story point the scrum master writes all the user stories as an-JIRA backlog and assigns them to the developers. When a developer starts working on a backlog he/she move the backlog from **To do** to **In progress**. Eventually when the developer finishes his/her work, he or she will move the backlog to **Done**.

# 4 Implementation

## 4.1 CDR processing (Application prototyping)

Billing is one of the core areas of a BSS application. Billing is done by processing the call detail records generated from the charging system. CDRs are generated for every network event like call, voice message, text message or internet browsing. Charging system saves those CDRs in a file. When a certain size is reached or a certain period of time has passed, these files get archived and a new files get created. Creating a prototype application for processing these archived files was my first project in the company.

### 4.1.1 Algorithm Flowchart

A proper algorithm design can only lead to a successful implementation of a project. An algorithm is a step by step method of solving a problem or doing a task. In other words an algorithm is a sequence of unambiguous instruction to solve a problem. On the other hand, a flowchart is a traditional graphical tool with standardized symbols. It represents the sequence of steps in an algorithm.

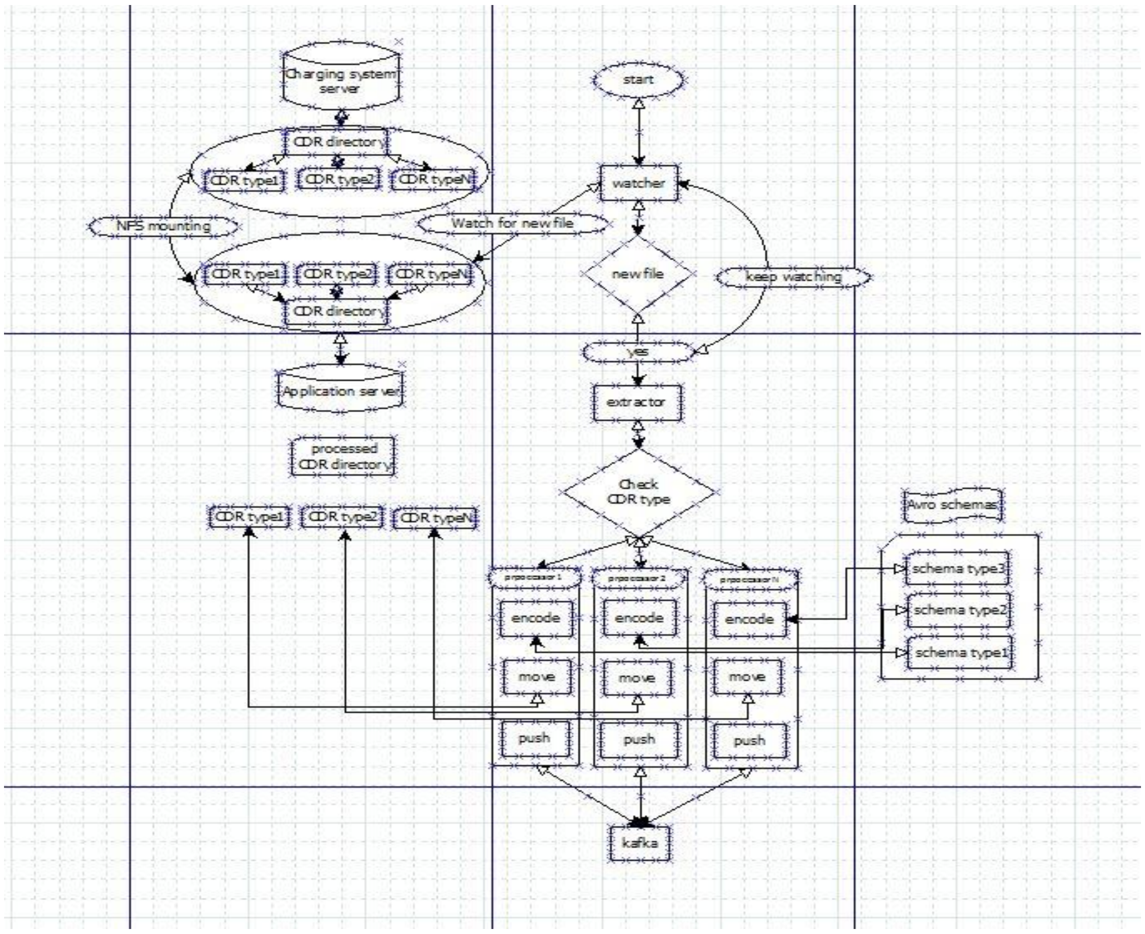


Figure 8: CDR processor algorithm.

When processing the CDRs I followed the above algorithm.

### 4.1.2 Network File System Mounting

In our company's integration setup thecharging system was running on a different server than the server I was supposed to use for my application deployment. To get the CDRs immediately to the server where my application was running I had to share the folder where the CDRs were getting generated between two servers. I used NFS mounting for accomplishing the job. NFS is a protocol that allows sharing file systems over the networks. Before starting mounting following steps are needed in the server from where the files will be shared.

First I had to install two software packages. The prerequisite software are **nfs-utils** and **nfs-utils-lib**. The NFS Utilities package contains the user specification and client tools necessary to use the kernel's NFS abilities. The **nfs-utils-lib** package contains support libraries required by programs in the **nfs-utils**package. Both of these software are available in the Linux yum repository and can be installed through the following command.

```
yum install nfs-utilsnfs-utils-lib
```

Subsequently I ran the following command to start NFS as a service and to start the **rpcbind** which is needed for translating the **rpc** program numbers into a universal address.

```
chkconfignfs on
servicerpcbind start
servicenfs start
```

In the next step I exported the file to the client server. The directory we wanted to share needed to be added to the **/etc/exports** file. It specifies both the directory to be shared and how it is shared. For sharing the home directory the following line needed to be added.

```
/home
<client-server-ip-address>(rw,sync,no_root_squash,no_subtree_check)
```

These settings are included for accomplishing the tasks below:

- **rw** gives the right to both server and client to write in the shared directory.
- **sync** notify the shared directory only once if any change is happened.
- If the shared directory is a subdirectory, NFS check the above directory for permission if **no_subtree_check**not mentioned. By mentioning this option I stopped the sub-tree checking. It increase the reliability of NFS mounting.
- **no_root_squash** enables root to connect to the specified directory.

Once the above procedures are done the following command will exports the file to client.

```
exportfs–a
```

For setting up the client, which in my case was my application server I had to install the same software **nfs-utils** and **nfs-utis-lib** in the client server. Once the installation has been completed the mounting can be done by executing the following command in the Linux terminal.

```
mount<host-server-ip-address>:/home /mnt/nfs/home
```

In this way the home directory of the host server in my case the charging system server can be shared with the /mnt/nfs/home folder of the client server. Any changes made on those folder will be replicated to other.

### 4.1.3 App Engine

The CDR processoris using theNodeJS and Avro serialization framework. The application extensively depends on the public node module to perform its operation. It starts with checking the CDR directory if there is any CDR to process. If there is no CDR available, it starts watching for file creating in the specified directory, in this case CDR directory. If there is any CDR left,it will uses the node module **node-dir**to list files. **node-dir**is a lightweight Node.js module with methods for some common directory and file operations, including asynchronous, non-blocking methods for recursively getting an array of files, subdirectories, or both, and methods for recursively, sequentially reading and processing the contents of the files in a directory and its subdirectories, with several options available for added flexibility if needed.**node-dir** returns all the files in anarray through its callback function. Thereafter the application goes through each and every file and pushes it for further processing and then starts watching for new file creation in the specified directory.

For watching the file creation event the application uses the node module **Chokidar**.**Chokidar** is a neat wrapper around the Node.js file watching module **fs.watch** or **fs.watchFile**. Although it uses the node module **fs.watch** or **fs.watchFile,** it solves some problem of those modules like reporting the same event twice. **Chokidar** emits events for all kinds of file or directory operations such as add, delete and rename. In my application I onlyused the add event to find out when a new file is added to the directory or to any of its sub directories.

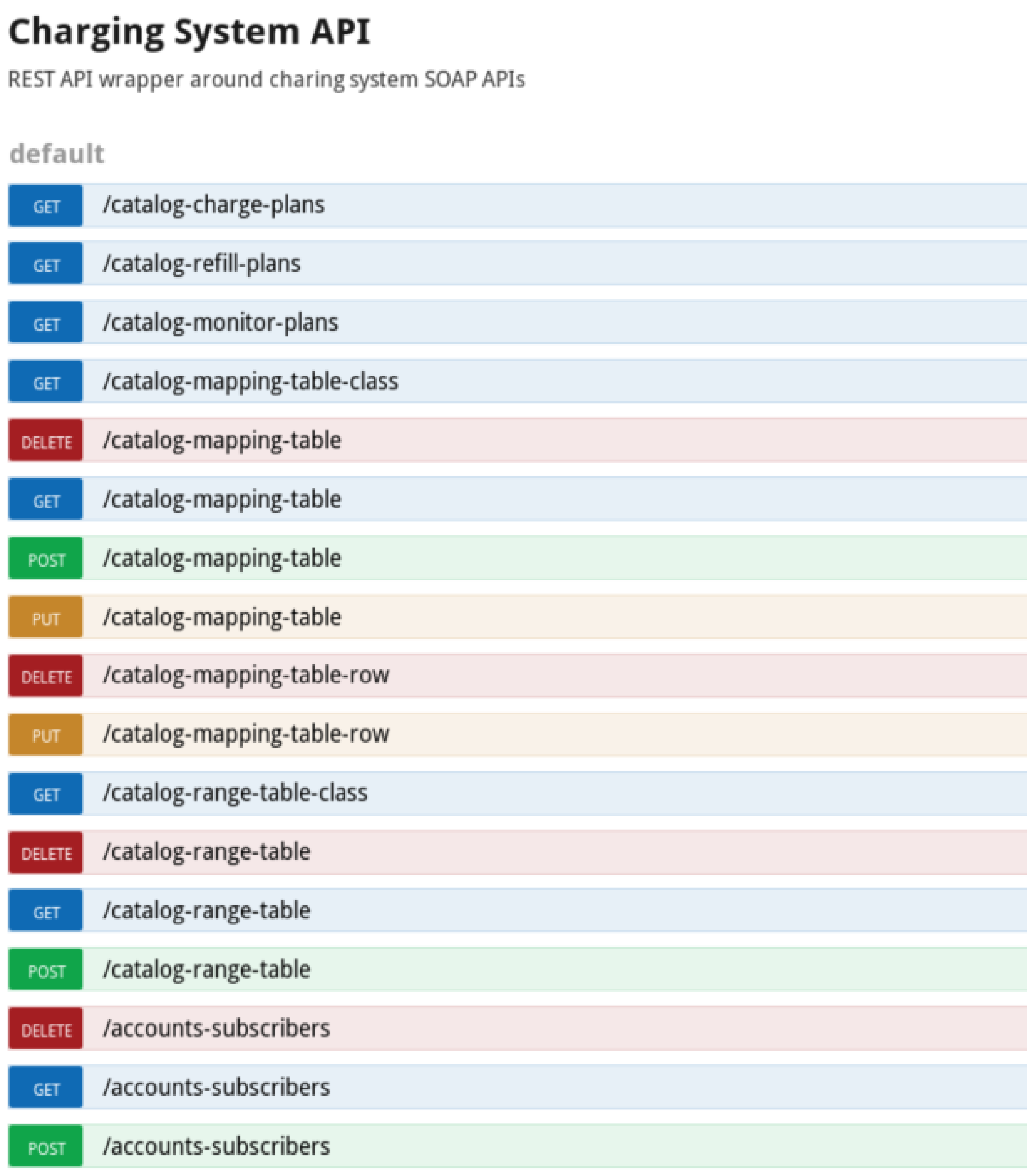
Charging system creates a GZ file which is a GNU zipped archive file. When a new file is created,**Chokidar**catches the file creation event and gives the path of the file through the callback function parameter. After I get the file path I use the node module **zlib**to decode the file. **zlib** is a data compression and decompression library provided by Node.js. This provides bindings to Gzip/Gunzip, Deflate/Inflate, and DeflateRaw/InflateRaw classes.I used **Gunzip** to decode the gz file. Decoding gives me a CSV format file. **CSV** file stores tabular data (numbers and text) in plain text. Each line of the file is a data record that consists of one or more fields, separated by commas.At that point I parsed the name of the file from the path string I got from **Chokidar**. These CDR files have a unique identifier in their name which defines their type. By using a regular expression I extracted the identifier from the name and used it for further processing.

In the next phase I used the identifier to decide which processor engine I should use to process the file and push the CSV file into the relevant processor.  The processor takes the CSV file as input and uses the node module **line-reader** to read the file. **line-reader** is anasynchronous line-by-line file reader. The **eachLine** function reads the file line by line. After reading each line the callback function gets called with two parameters, the data read from the line and a Boolean value which specifies whether it was the last line or not. If the Boolean value is false, then the **eachLine** function stops reading the file and closes it. The**eachline** function returns the line as a string. Later I parsed the string and parsing gave me an array of fields. Thereupon I mapped the fields into the JSON schema defined for Avro encoding.  Having done the encoding, I serializedthe schema using the node module **avro-serializer**. **avro-serializer**is a small library providing serialization and de-serialization routines for Apache Avro binary encoding.After serialization I stored all serialized data into an array.

Having read the entire file I pushed the array to Kafka for billing. If the push result was successful, then I move the file into a different directory and markedthe file as already read.

## 4.2 SOAP to REST Conversion

Our convergent charging system provides SOAP API to communicate with its database. As we were using REST API to communicate between the modules, I was assigned to convert the SOAP API into REST. To do that I wrote a Node.JS REST wrapper on the soap API. I used Swagger to give the API a nice UI and to document it. The API UI looks as shown in the following.



Charging System API

REST API wrapper around charing system SOAP APIs

default

```
GET     /catalog-charge-plans
GET     /catalog-refill-plans
GET     /catalog-monitor-plans
GET     /catalog-mapping-table-class
DELETE  /catalog-mapping-table
GET     /catalog-mapping-table
POST    /catalog-mapping-table
PUT     /catalog-mapping-table
DELETE  /catalog-mapping-table-row
PUT     /catalog-mapping-table-row
GET     /catalog-range-table-class
DELETE  /catalog-range-table
GET     /catalog-range-table
POST    /catalog-range-table
DELETE  /accounts-subscribers
GET     /accounts-subscribers
POST    /accounts-subscribers
```

Figure 9: Example charging system REST API.

I converted more than 30 APIs from SOAP into REST until now. There are some more APIs provided by our charging system that might need to be converted. All kind of modification tocharging system can be done through these APIs.

The name and details of the example APIs are thefollowing.

**GET /catalog-charge-plan** API returnsthe charge plans from the charging system database. If an id of the plan is given, it will return the specific charge plan.Otherwise it returns all the charge plans in the database. Charge plans are used to charge the customers.

**GET /catalog-regill-plan** API returns the refill plans from the charging system database. If an id is given as query parameter, the API returns only the specific refill plan otherwise it returns all the refill plans. Refill plans are used the refill the prepaid customers.

A monitoring plan class is the signature of a monitoring plan that defines how it is possible to configure the monitoring of a customer spending. **GET /catalog-monitor-plans** API fetches monitor plans from the charging system database. If an id of the plan is given then the specific plan is fetched. Otherwise all the monitor plans are fetched from the database.

A mapping table class is an assembly of the characteristics that define a group of mapping tables or subscriber mapping tables with the same structure. This master data is part of the pricing catalogue owned by a service provider. **GET /catalog-mapping-table-class** API fetches mapping table classes from the database. If an id is provided if fetches only one mapping table class with the specific id. Otherwise it fetches all the mapping table classes from the database.

A mapping table is a business data table of correspondence for mapping an input set of values to a set of output values according to different periods of time. This master data is part of the pricing catalogue owned by a service provider and is used to configure the pricing logic in the charging system.**POST /catalog-mapping-table** API creates mapping table in the charging system database. It takes the JSON payload as request body and creates a mapping table in the database.

**PUT /catalog-mapping-table** API updates the mapping table if needed. It takes the JSON payload as the request body and make the changes in the charging system database.

**GET /catalog-mapping-table** API fetches a mapping table data from the charging system database. If an id is given, then it fetches only a mapping table with the specific id. Otherwise it fetches all the mapping tables from the database.

**DELETE /catalog-mapping-table** API deletes mapping table from the database. It takes the id of the mapping table as query parameter and deletes it from the charging system database.

**PUT /catalog-mapping-table-row** API updates a specific row of the mapping table. Each row has an id. When the payload is sent it queries the specific row with the id and changes it in the charging system database.

**DELETE /catalog-mapping-table-row** API deletes a row from the mapping table rows. The API takes the id of the row as a query parameter and deletes it from the database.

A range table class is an assembly of characteristics that define a group of range tables with the same structure. This master data is part of the pricing catalogue owned by a service provider. **GET /catalog-range-table-class** API fetches range table classes from charging system database. If an id is given it fetches only the specific range table class. Otherwise it fetches all the range table classes available in the database.

A subscriber range table is a range table assigned to a subscriber account to be shared by the charging contract related to this subscriber account. A subscriber range table is part of the master data related to the end customers of the service provider.**POST /catalog/rangetable**API creates range table in the charging systemdatabase. It takes JSON payload as the request body and stores it in the charging system database.

**GET /catalog-range-table** API fetches the range table from the charging system database. It takes the id as query parameter. If the id is given then it fetches only one range table. Otherwise it fetches all range table from the database.

**DELETE /catalog-range-table** deletes the range table from the database. It takes the id of the range table as a query parameter and deletes it from the database.

Each subscriber has his or her own account in the charging system. All the information related to a subscriber is stored in his or her account. **POST /accounts-subscribers** API is used to create subscriber accounts in the charging system. It takes JSON data as the request body and creates the account in the charging system.

**GET /accounts-subscribers** API fetches the subscriber related data from the charging system database. It takes the id as a query parameter and fetches the specific subscriber from the database. If no id is given, then it fetches all the subscriber accounts from the database.

**PUT /accounts-subscribers** API works as a bundle or mass operation. More than one subscriber account can be updated with this API call. It takes an array of subscriber accounts as request body and change the subscriber accounts in the charging system-database by matching with the id.

**DELETE /accounts-subscriber** API deletes the subscriber account from the database. It takes the id of the account as an input and deletes the specific subscriber account from the database although in telecommunication, once a subscriber account is created, it will never get deleted.

**POST /accounts-external-account** API creates post-paid account in the charging system database. Post-paid accounts are called external account in our charging system hence the name of the API as above. The API takes the post-paid account related data as request body and creates a post-paid account in the charging system. All the post-paid account are attached to a subscriber account.

**GET /account-external-account** API takes the subscriber account id and the post-paid account id as input value and fetches the specific post-paid account.

**PUT /accounts-external-account** API update post-paid account in a bundle. It takes an array of external accounts as input and make changes in the charging system database by matching the post-paid accounts with the ids.

**DELETE /accounts-external-account** API deletesa post-paid account from the charging systemdatabase. It takes the id of the prepaid account and subscriber account as input value and deletes the specific account.

An allowance is a credit given to (or purchased by) a customer to use one or more services from a service provider. It can be defined as a quantity of a service or as an amount of money. An allowance can be valid for a specific period of time, for example:

10 MB of GPRS data valid for 10 days

5 Euros valid until the end of the month

**GET /post-paid-allowance** API fetches the allowances for one subscriber. It takes the id of the subscriber as input value and fetches the allowance given to that subscriber. Allowances are mostly used for post-paid subscribers.

**POST /accounts-prepaid-subscribers** API is used to create prepaid subscriber in the charging system. It takes JSON data as request body and create a prepaid subscriber in the charging system. Each prepaid subscriber is attached to a subscriber account.

**PUT /accounts-prepaid-subscribers** updates a bundle of prepaid account at once. It takes an array of prepaid subscribers as request body and updates the prepaid account by matching with the id.

**GET /accounts-prepaid-subscribers** API fetches the prepaid subscribers from the database. It takes subscriber account id and prepaid account id as input value and fetches the prepaid account attached to the ids.

**DELETE /accounts-prepaid-subscriber** API is used to delete a prepaid account from the charging system. It takes the id of the prepaid account and the subscriber account as input value and deletes the specific prepaid account.

Prepaid accounts has four states: active, blocked, locked and closed. State of the prepaid account can be changed with **PUT /accounts-prepaid-account-state** API. It takes subscriber account id, prepaid account id, the state and the date as input and make the changes in the charging system database. If the state is already closed, it cannot be reverted.

**POST /prepaid-account** API refills the prepaid account. It takes the prepaid account id and the refill amount as input value and does the refilling in the database.

**GET /prepaid-account-from-user-technical-identifier** API fetches the prepaid account with MSISDN number.MSISDN isan acronym for Mobile Station International Subscriber Directory Number. This is the number people dial when connecting to another phone.  It takes the MSISDN number as input and fetches the prepaid account from database.

A charging contract is the pricing and charging view of a provider contract stored in the charging system. This master data relates to a long-term business relationships between an end-customer of a marketable service and the service provider.**POST /charge-contracts** API is used to create charging contract in the charging system. It takes charging contract related data as JSON input and creates charging contract in charging system database. Each charging contract is bound to a subscriber account.

**PUT /charge-contracts** API updates charging contracts in bundle. It takes an array of charging contracts as input value and update the charging contracts by matching with id.

**GET /charge-contracts** API is for fetching the charging contract from the charging system database. It takes charging contract id, subscriber account id and charging contract state as input and search the specific contract from the database. If only charging contract id is given, it fetches the charging contract with the specific id. If only subscriber account id is given, then it fetches all the charging contracts for that subscriber account. If only the state is given all the charging contracts with the state are fetched from the database.These parameters can be used in any combination. If no parameter is given, then the API fetches all the contracts in the database.

Contract has three states: active, locked and closed. **PUT /charge-contract-state** API changes the contract state. It takes charging contract id and the state as input and changes the contract state in the charging system database accordingly. If the state of the contract is closed that cannot be reverted.

**DELETE /charge-contracts** API deletesa charging contract from the charging system database. It takes the id of the charging contract and deletes it from the database.

**POST /customer-data** is an atomic API and actually built for transferring data from one service provider to another. However create operation is also possible if no data with the same id is available in the charging system database. It is possible to create subscriber account, prepaid account, external account, contract and allowance with this API. Although the operation of the API is very useful, the performance is not as good as other APIs.

**GET /customer-data** API fetches all the data related to a specific customer. It takes the subscriber id as input and fetches the data specific to the subscriber.

**POST /charge-subscriber** API is used to simulate charges for the subscribers. It takes JSON data as input and creates a charging CDR for the customer in case of the postpaid subscriber.For prepaid account it deducts money from the balance. Charge for calling to a number or sending a text message or using the internet can be simulated with this API.

**POST /charge-mass-bundle-subscriber** API also creates charges for the subscriber. However it does it in a bundle. It takes the array of charge-related data and simulates a charging event.

## 4.3 Order Management

We have used node micro service approach to build the order management project and pm2 to manage it. In micro service approach small Node applications are loosely bound with each other using some message queues. In our case it is Kafka messagequeue. Figure 10 shows the architecture of the order management project.

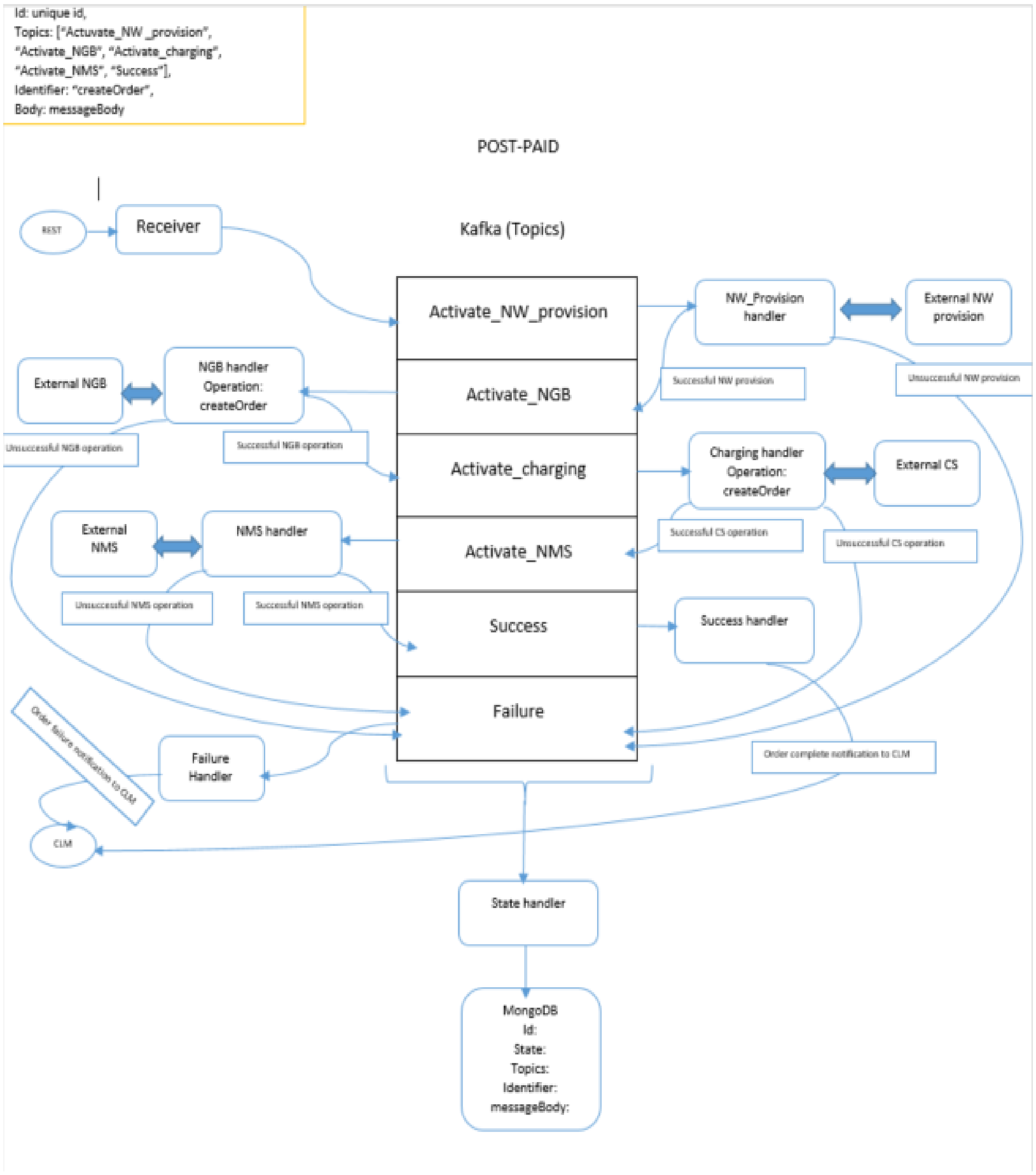
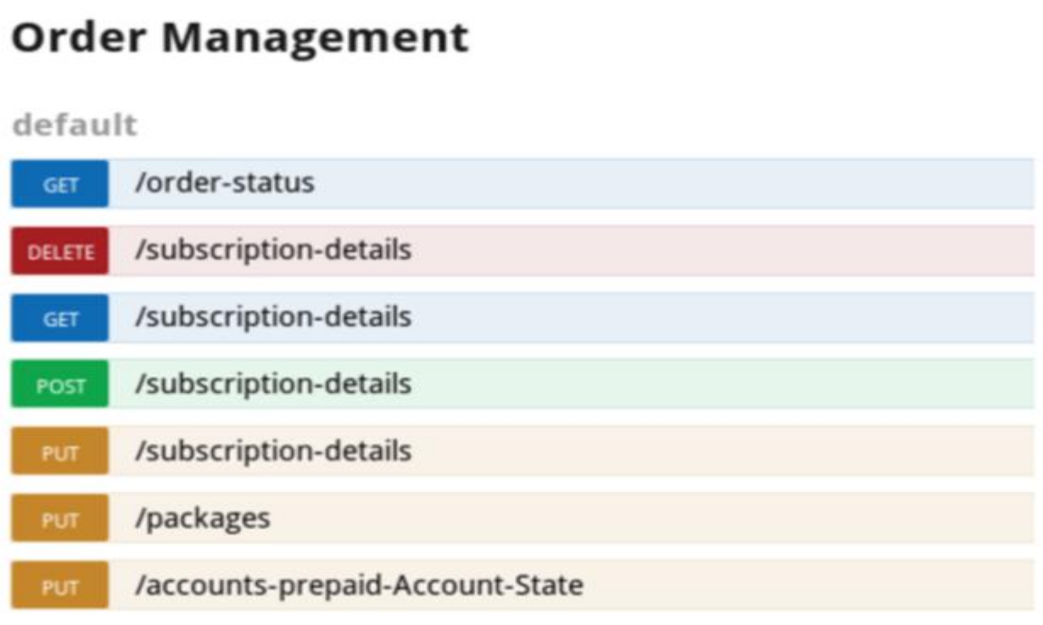


Figure 10: Order management architecture.

The Receiver in our case, which is Order Acceptance module takes the order from client lifecycle management module (CLM)/ (CRM) and then pushes it to a specific topic in the Kafka. The next module already listening to the topic receives the message processes it and sends it to the next module. This flow of action is not sameevery time it is operation-specific. The flow of action is set at the first module based on the operation.

The order management module offers REST API to CLM and the REST API is documented using swagger.


Order Management

default

GET /order-status

DELETE /subscription-details

GET /subscription-details

POST /subscription-details

PUT /subscription-details

PUT /packages

PUT /accounts-prepaid-Account-State

Figure 11: Example of Order management REST APIs.

Figure 11 shows the example API offered by Order Management system.

### 4.3.1 API specifications

The name and details of the API that order management system offers to CLM are following.

The order management module works in an asynchronous way. It takes the order from CLM and immediately returns an order id to CLM and start its operation. When the operation is done if the result is successful then it sends a success message to CLM however if the operation fails it creates a trouble ticket and send it to trouble ticket server.**GET /order-status** API is used to track any order. It takes order id or customer id as query parameter and return the status of the order. If it is successfully completed or still stuck in some module or failed.

**DELETE /subscription-details** API is used to terminate a subscription. It takes the id of the subscriber and terminate the subscriber from network provision system, billing system, charging system and number management system.

**GET /subscription-details** API is for fetching all the data about the subscriber from backend servers. It takes the subscriber id as query parameter and fetches the data about the specific subscriber.

**POST /subscription-details** API takes JSON data about subscriber details as input and does all the backend operation such as network provisioning, creating an account in billing system, creating an account in charging system, and reserves a number in number management system to create a new subscriber.

**PUT /subscription-details** API suspend or revoke subscription. The type of operation can be specified with a parameter. It takes the ids of the attached accounts and charging contract to the subscriber account as input and suspends or revokes the subscription from all the backend servers.

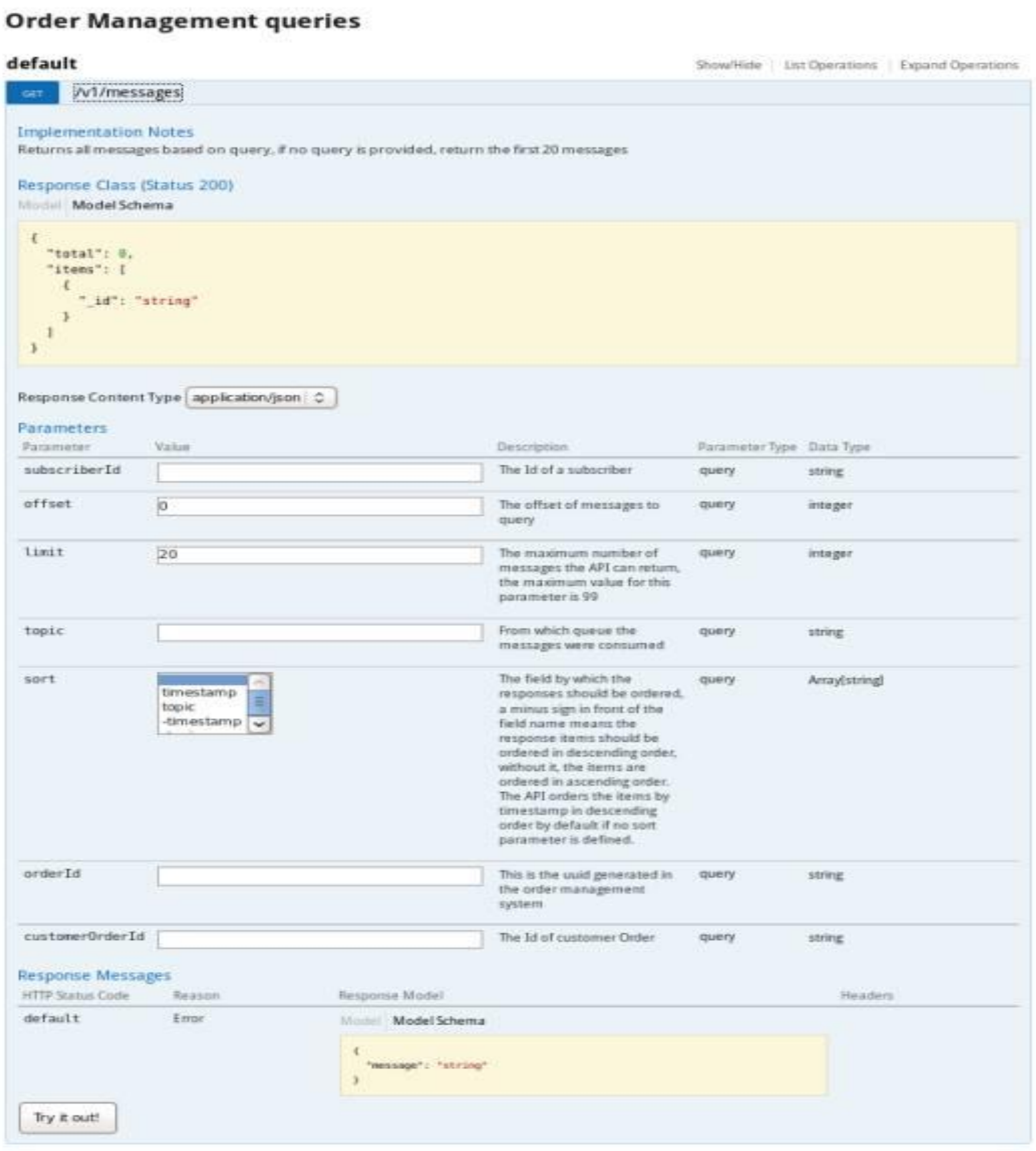
**PUT /packages** is used to change the offered packages for a subscriber. If the subscriber wants to change his or her package, then this API will take the new offer-related data and make the change in charging system.

**PUT /accounts-prepaid-Account-State** API is for changing the state of the prepaid account into active from blocked and locked and vice versa. It takes the prepaid account id, the subscriber account id and state as input and changes the state in backend systems.

**PUT/accounts-postpaid-AccountState** API is for changing the state of the post-paid account to non-active to active and active to terminate. It takes the post-paid account id, the subscriber account id, service account id and date as input and changes the state in backend systems. Once a post-paid account is terminated it cannot be reverted.

### 4.3.2 Micro services

**Persistence module** is used for saving all the orders and their states. Furthermore we are also using it for recovering from crush. In Figure10 it is defined as state handler. It is a standalone module and does not participate in the order management process. It listens to all the topic in the Kafka and consumes the messages comes to thesetopics. It exposes a REST API through which it is possible to query an order status by order id, or customer id. Furthermore the orders can also be queried by Kafka topic and subscriber account id. We have used Node.JS express framework to build the application, swagger to document the REST API, Mongodb to store the data and node module pm2 to manage the application. Figure 12 shows the swagger UI of persistence module.



Implementation Notes

Returns all messages based on query, if no query is provided, return the first 20 messages

Response Class (Status 200)

Model | Model Schema

```
{
  "total": 0,
  "items": [
    {
      "_id": "string"
    }
  ]
}
```

Response Content Type: application/json

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| subscriberId | | The Id of a subscriber | query | string |
| offset | 0 | The offset of messages to query | query | integer |
| limit | 20 | The maximum number of messages the API can return, the maximum value for this parameter is 99 | query | integer |
| topic | | From which queue the messages were consumed | query | string |
| sort | timestamp<br>topic<br>-timestamp | The field by which the responses should be ordered, a minus sign in front of the field name means the response items should be ordered in descending order, without it, the items are ordered in ascending order. The API orders the items by timestamp in descending order by default if no sort parameter is defined. | query | Array[string] |
| orderId | | This is the uuid generated in the order management system | query | string |
| customerOrderId | | The Id of customer Order | query | string |

Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|---|---|---|---|
| default | Error | Model \| Model Schema<br>{ "message": "string" } | |

Figure 12: Persistence module Swagger UI.

Each operation in order management has a specific topic for each module. When the work of one module gets finished, the order is pushed to the next topic in Kafka queue.Then the module listening to that topic starts processing the order. Persistence module consumes the message when it placed in Kafka for the first time and pushes it to database with its topic name which is the state of the message. In the order management queue when the message gets processed and pushed to next module the persistence module listens the same message from another topic. It updates the topic off the message to new one hence the state of the message is also get updated. In this way each message eventually ends up in success of failure state.

**Order Acceptance module** is the first module that directly communicates with the CLM. It exposes the REST API for CLM operation. I have used Node.JS express framework to build the module and swagger to document the REST API. When it receives a request from the CLM, it validates the request data. The validation is done by a node module I have created. It is possible to download the module from git by declaring the path in node **package.json** file. The reason behind not including it with the main module is to give more modularity. In the future if the backend systems change we will not have to touch the main module for validation.

The validation can be turned off by specifying a parameter to off. If the validation fails, it returns "HTTP 400 Bad request" message to CLM. If it passes, then the module formats the data for further operation. Formation includes ascending the attributes in order if they are not and making separate object for different operation. After formation being done, the module addsa unique id and array of Kafka topics to the message. The unique id is the order id. The array of Kafka topics defines the flow of action for any specific operation. Finally the module pushes the data to Kafka queue and updates the UI. We have created a UI for testing purposes. This UI is updated when the message is placed and when the message reaches the success or failure state.

**Network provision handler module** is second module in the queue. It does the network provisioning for the subscribers. Provisioning is needed to give the user access to the telecommunication network. Thismodule startswith querying the database if there is any order that is not yet processed. This feature is implemented for crash recovery. If a request comes after the server gets crashed, it will remain in the database until processed. The persistence module consumes all the messages and saves them in the database with their topic name and Kafka offset.  After querying the database the module start the consumer from the point it left processing. For implementing this I have used offset commit feature of Kafka.After fetching the objects from database the application checks for the Kafka offset of the first object and commit it to Kafka. Finally it starts the consumer. As the offset of the first message that is not yet processed gets committed to Kafka, the consumer starts from that specific offset and hence each and every order gets processed.

All the operations in order management does not need network provision.  This module only listen to the topic where orders that needs network provisioning comes. In other word Order Acceptance module only sends those messages that needs network provisioning to those topics where Network provisioning handler is listening to. After consuming the message it checks for the topic of the message to determine its operation. Our company has a network provisioning server simulator and we are using that to do test integration of network provisioning system with our product. Our provisioning server expose a SOAP API for provisioning operation. It takes information like IMSI, MSISDN, SIM number, switch number and etc. for performing its operation.

All this information comes from CLM. My work in this module is to map those parameters into a SOAP request and make the request to provisioning server. Telecommunication operators normally use their own provisioning server provided by giant companies like Ericson or Huawei. Hence network provisioning integration is normally done at the time of deployment. For making the integration easier, we have used the loosely bound architecture so that if we need to change the network provision handler, we would be able to do it by not affecting the other part.

**Billing handler module** is third module in order management project and does the integration with the billing server. It is used to perform any operation attached to post-paid accounts as prepaid accounts do not need billing. Four types of operations are done with this module to billing server. The operations are to activate a post-paid account, to terminate a post-paid account, to pause a post-paid account and to resume a post-paid account. Our billing server exposes REST API for performing those operations. For performing these operations the billing server needs some information like post-paid account id, subscriber id and date.OM gets all this data from CLM. The work in this module is map those data into a REST request and make request to billing server. I used Node module request.js for sending request to billing server.

This module starts with checking the database as the network provisioning handler module does. It queries the database, push the offset of the orders that are not yet processed and start the consumer. When the consumer consumes a message, it checks for the topic of the consumer and does the operation accordingly. For example the terminate subscription operation comes to the topic called **NGB-Terminate-Subs**. In consumer if the message topic matches with **NGB-Terminate-Subs**it forwards the message to do the termination of a post-paid account in billing server. When the operation is done, it adds the result of the operation to the request body and send it to the next module. If the operation fails, then it sends the data to failure handler.

The fourth module in the order management project is **Charging System handler module** and it does the integration with the charging system. The charging system provides a SOAP API for any kind of operation with its backend server however in my last project I have created a REST wrapper on that SOAP API hence I used that for making request to charging system backend server.

This module performs all types of operation with the charging system. It creates, activates, blocks, unblocks, locks, unlocks, closes and changes subscriber account and charging contract in chargingsystem. WhenKafka consumer consumes a message from **Kafka,** like the other module it also determined the operation by checking the topic of the message and then perform the operation. After performing the operation it add the result of the operation with the request data and send it to next module however if it fails to perform the operation it sends the order to failure handler.

**Number management system handler module:**This module is the fifth module in the order management project and does the integration work with the number management system server. The number management server is used to manage the MSISDN numbers. The MSISDN numbers are unique numbers. Hence once a MSISDN number is used it needs to be reserved. Number management system does the reserving and releasing work. It also does the SIM blocking and unblocking operation. Our number management server exposes an API for these operations which take xml input and does its operation. For performing these operations the NMS server needs the MSISDN number and the SIM number. All these information comes from CLM. The work of these module is to map the JSON data into an XML request and make the request.

Like other modules, it also starts with querying the database and committing the Kafka offset if necessary. When a new message arrives at the consumer, it checks the topic of the message and determines the operation accordingly. If the operation succeeds it will push the message to the next topic. In our case it is the success topic in Kafka. However if it fails, it will go to failure.

**Success failure handler** is the last module in the order management project. It consumes messages from **success** and **failure** topics. When a new message comes, it will check for the topic of the message. If the topic is **failure,** it will generate a trouble ticket and send it to the CLM trouble ticket server for human interaction about that order and finally update the test UI we have created for simulate the CLM UI. However if the topic is **success** it only update the CLM UI with a success message.

For creating the trouble ticket CLM exposes a REST API. In this module I create a JSON object with required data which includes the error message, customer id,and subscriber id and sends it to the CLM trouble ticket server for further processing.

# 5 Results and discussion

The project is not completed yet, hence it is difficult to give specific results of the project yet. However the modules that are already completed gave promising results compared with the other software running in the industry. As per my responsibility in the project, I have completed it successfully. I was able to follow the roadmap designed for the work I was doing and the results are stable. The continuous integration and testing with Jenkins are passing without any problems. However it needs to pass a few more labels of testing before deployment.

The prototype of call detail record processing engine I have crated can process 1200 CDR entries per second in a test setup that is running on a virtual machine and has anCentOS operating system, Intel core i5 processor and four GB main memories. This is the output when I tried to process one million CDR entries. This performance is good in a system like this and compared to a Java-based CDR processing engine. We will be able to boost the performance by increasing the system configuration at the time of deployment. It is also possible to start the CDR processing engine in the cluster mode using node module **pm2**. In this case more than one CDR processing engine will process the CDR at the same time.In this way it is possible to multiply the performance of the CDR processor.

The development of Order management project has been progressing in target schedule. The biggest risk in the beginning of the project was to implement the Node micro service architecture and we have successfully completed it. All the features we have completed until now are stable and giving positive results when testing with Jenkins. For making it a production-grade application we need to add some more APIsand more tests.

The implementation of the REST API layer on charging system provided SOAP API was also a successful project. It has been used by **unified product catalogue** module, **client lifecycle management** module and **order management** module for integration without any problems. The API layer is running without any problems since its first deployment at the end of May 2015 although it was deployed very rapidly.

Today's telecom operators are trying to change their business strategy by giving less emphasis on thelegacy product and service and more on the Internet-based service. Internet based services like Netflix or Spotify are becoming very important catalysts for getting customer attention. Hence they are trying to move to new generation BSS software which will give them the ability and configurability to integrate these services into their business while keeping their legacy product as it is. We are trying to give them the new generation BSS they are looking for.

# 6 Conclusion

The goal of the project was to create new generation BSS software for the service operator to adopt the change in their business. The project is not completed yet. However we are approaching the landmark gradually by following the roadmaps.We are hoping to attract new customers with our flexible and modern product suite as well as making our ground strong in the places where we already have customers.

My personal goal was to become a competent Node.JS programmer when I started in the company and I think I am successful on that. I have implemented two application in Node.JS which has given positive results at the time of testing. Furthermore I am working as the lead developer in the order management project. The project is not finished yet. However what has been implemented so far is giving reliable and responsive results.

The telecommunication sector has changed considerably in the past decade and it is changing at a supersonic speed. Hence the needs of the telecom service provider are changing with that pace. They want something, they want it now, they want it to be robust and they want it to be cheap. We have addressed all these requirements since we started developing this product. We are confident to deliver a product that will be good in performance as we are using very new technologies when developing the product.The project was started keeping configurability in mind. So the application will be able to adopt any change very easily. In the case of adding new servicewe will not have to touch the core product but we will be able to do it by adding a new module to the existing product.

## References

1. Ericsson. Ericsson dynamic catalog [online] . Piscataway, NJ 08858 USA: Ericsson; 2012.URLhttp://archive.ericsson.net/service/internet/picov/get?DocNo=1/28701-FGB1010129&Lang=EN&HighestFree=Y. Accessed 21 July 2015.

2. TATA. TCS Hosted OSS/BSS CRM Solution [online]. India: TATA; March 2014. URL:http://www.tcs.com/SiteCollectionDocuments/Brochures/Hosted-OSS-BSS-Customer-Relationship-Management-0414-1.pdf. Accessed 25 July 2015.

3.  Apache. Apache Kafka A high-throughput distributed messaging system. Apache; URL:http://kafka.apache.org/. Accessed 12 August 2015.

4. Node.JS foundation. Home [online] .Portland, USA. Node.js foundation;2015. URL:https.://nodejs.org/en/.Accessed 13 August 2015.

5. The Swagger community. Getting Started [online] . 450 Artisam Way, Somerville MA 02145, USA:Smartear; 2015.URL:http://swagger.io/getting-started/. Accessed 28 August 2015.

6. The Apache Software Foundation. Apache Avro 1.7.7 Documentation[onlinne] . Maryland USA:  Apache Foundation; 24th July 2014. URL:http://avro.apache.org/docs/current/. Accessed 20 September 2015.

7.  Red Hat Inc. Red Hat Cluster Suite Introduction[online] .Releigh, North Carolina, USA: Red Hat Inc; 2015.URL:https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Cluster_Suite_Overview/s1-rhcs-intro-CSO.html. Accessed 22 September 2015.

8. Akemi Yagi. About CentOS[online] . 17 September 2015.URL:https://wiki.centos.org/. Accessed 24 September 2015.

9. Atlassian. Jira Software overview [online].London United Kingdom: Atlassian; URL:https://confluence.atlassian.com/jirasoftwarecloud/jira-software-overview-779293724.html. Accessed 28 September 2015.

10.Florian Haupt et al. A model-driven approach for REST compliant services. Universitätsstr. 38, 70569 Stuttgart, Germany: Institute of Architecture of Application Systems university of Stuttgart; 2014.

11.DiomidilSpinellis.  Git. Athens Greece: IEEE; 2012.

12. Jon Skinner. Some things user love about Sublime Text[online] . Proprietary software.URL:http://www.sublimetext.com/. Accessed 21 October 2015.

13. Tony Gorschek. Third International Workshop on Software Product Managent - IWSPM'09. ACM;2010.

14. SohagSharkar. Billing
Solution for Telecom Sector. Pune, India: Institute Teleco Management Pune; 2004.

15. Ernst & Young. Global Revenue Assurance survey 2013.  EYGM Limited; 2013.

16. Atlassian. Meet Jenkins[Online]. Confluence; URL:https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins. Accessed 23rd October 2015.

17. Amani Mahdi Mohammed Hamed, HishamAbushama. Popular Agile Approaches in Software Development Review and Analysis. Khartoum, Sudan; IEEE. 2013.

18. Martin Fowler. Continuous Integration [online]. Martin Fowler; 2012. URL: http://www.martinfowler.com/articles/continuousIntegration.html. Accessed 23rd October 2015.

19. Ravi Sharda. Telecom OSS/BSS: An overview. Ravi Sharda; 2010. URL: http://ravisharda.blogspot.fi/2010/03/telecom-ossbss-overview.html. Accessed 30 July 2015.