KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

AKSHAY RAGHUNATH AVHAD

INVESTIGATION OF IoT (INTERNET OF THINGS) APPROACH TO DATA EXCHANGE AND VISUALIZATION IN MES & ERP INTEGRATION

Supervisor
Prof. Gintaras Dervinis

KAUNAS, 2016

KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

INVESTIGATION OF IoT (INTERNET OF THINGS) APPROACH TO DATA EXCHANGE AND VISUALIZATION IN MES & ERP INTEGRATION

Final project for Master degree
621H66001 CONTROL TECHNOLOGIES

Supervisor
Prof. Gintaras Dervinis
(date)

Reviewer
Prof. Dr. Leonas Balaševičius
(date)

Project made by
Akshay Avhad
(date)

KAUNAS, 2016

KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

Akshay Avhad

621H66001 CONTROL TECHNOLOGIES

INVESTIGATION OF IoT (INTERNET OF THINGS) APPROACH TO DATA EXCHANGE AND VISUALIZATION IN MES & ERP INTEGRATION

DECLARATION OF ACADEMIC HONESTY

/ /2016

Kaunas

I confirm that a final project by me, **Akshay R. Avhad**, on the subject "INVESTIGATION OF IoT (INTERNET OF THINGS) APPROACH TO DATA EXCHANGE AND VISUALIZATION IN MES & ERP INTEGRATION" is written completely by myself; all provided data and research results are correct and obtained honestly. None of the parts of this thesis have been plagiarized from any printed or Internet sources, all direct and indirect quotations from other resources are indicated in literature references. No monetary amounts not provided for by law have been paid to anyone for this thesis.

I understand that in case of a resurfaced fact of dishonesty penalties will be applied to me according to the procedure effective at Kaunas University of Technology.

(name and surname filled in by hand) *(signature)*

Avhad A. Investigation of IoT (Internet of Things) Approach to Data Exchange and Visualization in context of MES & ERP Integration. *M.Sc Control Systems* final project / supervisor Prof. Dr. Gintaras Dervinis; Kaunas University of Technology, faculty of Electrical & Electronics Engineering, department of Automation.

Kaunas, 2016. XX p.

SUMMARY

This research study presents a methodology to optimize production process for manufacturing industries. An ethical and technical approach is suggested for information transfer between control or process layer and top layer business system. System Integration of ERP (Enterprise Resource planning) and Process layer is achieved through a key element called Manufacturing Execution system (MES). This article will enlighten upon how the enterprise resource in an overall success of resource optimization, production, and management.

Integration of MES and ERP is a key to information transfer in context of ISA-95 standards. Industry 4.0 focus about an implementation of IoT (Internet of Things) in smart manufacturing practices. In a global enterprise scenario, ERP systems are distributed over the world. ERP systems do not belong to just single enterprise, it could be a cluster of partner companies comprising of multiple suppliers and consumers. SaaS (Software as a Service) cloud architecture enables to offer fully developed application as a service to the client users. Multi-tenant architecture is analyzed to serve the needs of multi-enterprises scenario. Provider or supplier entity along with consumer client entity, are considered as tenant model on a company multi user scale. A prototype of such system was developed with a Grails framework based on java based on servlet API of *J2EE* and *SPRING* framework under the hood. A degree of development to support a described multi-tenant application has been evaluated in the Grails framework.

Big Data analytics also play a role in visualizations for optimization of the production process. An example of *SYSLAB* grid using an open source visualization library “D3.js” based on HTML5 and JavaScript using connected components algorithm is explained.

Keywords— *MES, ERP, System Integration, Business system, ISA 95, Industry 4.0, IoT, J2EE, SPRING, SYSLAB*

ACKNOWLEDGEMENT

I would first like to thank my thesis supervisor Prof. Gintaras Dervinis, Head of Department of Automation at Kaunas University of technology. Prof. Dervinis has always portrayed faith in me right from topic selection to choosing a path to carry out research flow. He consistently motivated me to steer in direction which was difficult for me to pursue unless a considerable amount of freedom had been granted.

I would like to express profound gratitude to Systec & Services GmbH, Karlsruhe, Germany for inviting me to work on this research topic. Special thanks to Mr Jens Lukas, Director Development, Systec & Services who always guided me through difficult implementations and concepts; and was always available whenever I ran into trouble spot or had a question about my research.

Akshay Avhad

Table of Contents

1. INTRODUCTION

Demand for manufacturing production has been ever increasing day by day with growing needs of the exploding population. Food production itself has to grow [1] by 70 % by 2050 than today to maintain supply need. Food processing industry has to indulge into optimal production to increase processed food. This is possible with efficiency in supply of raw material to the processing facility in order to avoid wastage; food industry is most prone to wastages due to inefficiency in value chain. Pharmaceutical industry for instance has biggest challenge of adopting serialization standards to overcome counterfeiting in market through exposing vulnerabilities in the supply chain from provider to consumer companies and eventually to the retail partners. In developing countries like India, this is an immense challenge in due to lack in economic freedom and lack of regulation with regards to need of manufacturing of domestic market.

Manufacturing Industry is a stack with layers consisting shop floor itself comprising of all sub production units, business executives who regulates the key decision depending on the market need, and a sandwiched layer which binds the lower factory floor to top enterprise system called as manufacturing execution system (MES). In context to single enterprise these 3 layers are sufficient to drive supply chain or value chain. Industries are already into globalization scenario where an enterprise may have production factory units placed all over the world and the executives operating through a central location. This makes the supply chain dispersed over the globe, with multiple providers/suppliers and multiple OEM consumers.

Internet of Things (IoT) has already been seen as driving force in manufacturing for future. The concept of bringing manufacturing information on internet leads is very important aspect in implementing smart manufacturing. Industry 4.0 is the term for the combination of smart manufacturing and IoT. Cloud computing is a hot topic especially in commercial IT applications, web application are proving efficient in sharing information through cloud servers. Manufacturing scenario too can incorporated with this technology, which will give an advantage in data exchange on global scale, transparency in manufacturing practices and modularity. Industry 4.0 talks about *Big Data* application in cloud computing and data analytics to form an intelligent manufacturing environment.

2. LITERATURE REVIEW

2.1 Enterprise Resource planning

Enterprise Resource Planning (ERP) [2] is such an important tool, which helps the company to gain competitive edge by integrating all business processes and optimizing the resources available. Companies that implement solutions that take advantage of these accepted standards can more easily integrate their disparate systems and gain better visibility and coordination between the enterprise and execution layers. Companies that have successfully integrated the ERP with the shop floor are realizing faster cycle times, higher throughput, better quality and improved decision making for a significant competitive advantage. Management needs to know about activity on the plant floor while manufacturing needs up-to-date information on demand changes to help develop and optimize their schedules. Information at the ERP gives manufacturers the vantage point to make the appropriate changes based on what is happening on the plant floor to improve the overall efficiency of the operation.

Application of ERP has benefitted manufacturing industry in following through:

1. Lower cost in total supply chain: Effective decision making on enterprise level has benefitted in reducing loss in production.
2. Shorten throughput times
3. Reduce stock to minimum: Reduction piling up stock due to integrated
4. Enlarge product assortment
5. Improve product quality
6. Provide more reliable delivery dates and higher services to consumer
7. Efficiently coordinate global demand, supply and production.

In the fig 2.1 below ERP is the Information top layer in the manufacturing pyramid, MES and production layer are more in manufacturing context.

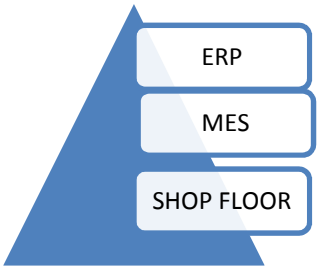

Fig 2.1shows manufacturing pyramid [30]

2.2 ISA model governing control & information system

ISA-88

ISA-88[3] model defines methodology for streamline and efficient production control through a uniform universal structure. Originally being developed for batch control process later has been implemented for continuous and discrete processes too. Basically s88 standard improves production process which is concerned with the layers 0, 1, 2 on manufacturing operational layer.

S88 defines workspace for recipe and equipment. Process engineers have responsibility of running the recipe while the control engineer has to make sure of the proper equipment operation. S88 defines the role of different sects in production. It provides guidelines for normal operation and abnormal condition exception handling in most cases comprises larger and critical part of program. S88 eases and clears the way for communication between vendors and customers.

Need for S88 standard because of following factors**:**

1. There was a lack of a universal model for batch control.
2. Users had difficulties in communicating their batch processing requirements.
3. Engineers found it very hard to develop integrated solutions with equipment of different vendors.
4. Engineers and end-users had configuration problems for batch solutions.

2.2.2 ISA 95

The ANSI/ISA-95 [4] standardizes data exchange between MES and ERP business layer and proves to be efficient model for integration between both the models.

ISA-95 is the international standard for the integration of enterprise and control systems. ISA-95 consists of models and terminology. These can be used to determine which information, has to be exchanged between systems for sales, finance and logistics and systems for production, maintenance and quality. This information is structured in UML models, which are the basis for the development of standard interfaces between ERP and MES systems. The ISA-95 standard can be used for several purposes, for example as a guide for the definition of user requirements, for the selection of MES suppliers and as a basis for the development of MES systems and databases. It standardizes the exchange of information between ERP systems and MES systems.

2.2.3 Integration of ISA-95 and ISA-88

From the discussion [5] on scope of influence of SA-88 and SA-95, it is very much clear that level - 0, 1 and 2 are defined with ISA-88. MES and Business layer are governed by the principles of ISA-95. Integration of MES and ERP is very much dependent on the handshaking of both the models on functional level as shown in Fig 2.2.

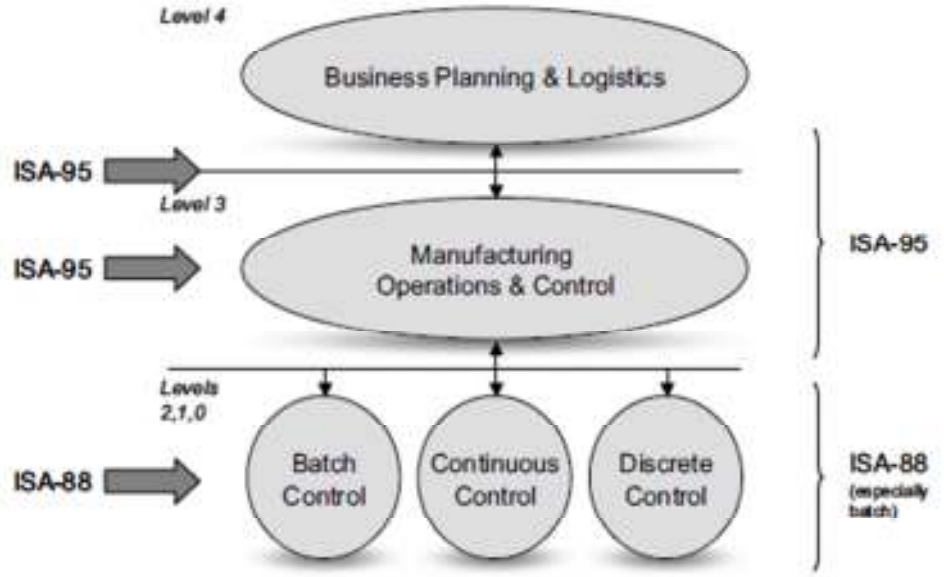

Fig 2.2 Illustrating integration boundaries for ISA-95 and ISA-88

To move in the direction of integration for MES, the need for specifications of ISA88 and ISA95 model has been compared in following table 2.1

Table 2.1 to evaluate difference between ISA-88 and ISA-95[27]

ISA-88 model / concept	**ISA-95 model / concept**
Physical model	Equipment Hierarchy model
Control Activity model	Activity model of Production Operations Management
General, Site, Master & Control Recipes	Product Definition Information & Product Definition Management

Batch Production Records	Production Performance Information & Production Data Collection & Tracking

2.3 Integrated and Flexible MES based on Agents and SOA

This article [6] introduces integration of MES based on Agency and SOA (Service Oriented Architecture) along with legacy (old) systems without compromising on interoperability of MES system.

Based on task agencies, MES layer can be categorized into 9 different agencies:

Fig 2.3 Illustrating software agents key [28]

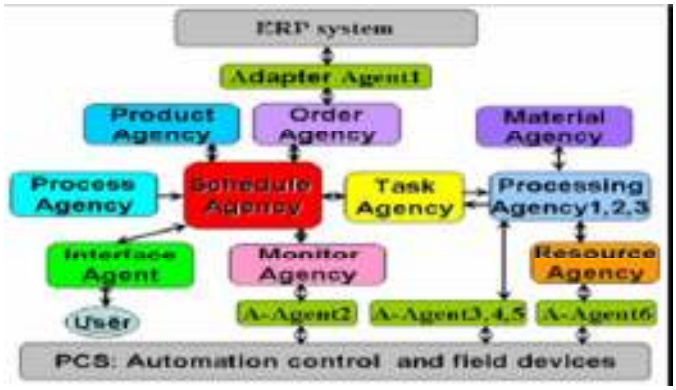

Fig 2.4 Shows integration solution using software as an agent [28]

All the Adapter agents are key components in integrated ERP, legacy systems operating as a whole system. This system provides an instant plug and play function. Efficient integration using an Adapter Agent is designed to perform the integration of legacy systems with MES. The Web Service is adopted to realize the communication between agents and the Market Mechanism is adopted to serve as control for interaction behaviour among agents.

2.4 Manufacturing Execution Systems for Sustainability

MES [7] has not been only looking towards conventional production issues but instead developed towards sustainable production too, and this paper introduces this specifically. The sustainable term was introduced in 1980 in OECD (Organization for Economic Cooperation and Development). The idea of this is to maintain production for present at the same time without compromising the needs of future energy. MES guides production process, initiates it and responds to critical plant activities. Apart for this MES has also a role in following factors:

1. Accuracy in data;
2. Significant resource management;
3. Information and planning capabilities to reduce waste;
4. Management of raw materials and resources;

Reduction in energy usage and other sustainable goals needs information from production chain in process to operational control system in order to provide energy integration. Resource utilization and efficient optimization are key factors in gaining sustainability. Also finding goals on plant specific, industry specific in reducing waste of valuable raw materials is important aspect.

Sustainability in MES can be achieved by implementing following points:

1. Automation of process lines.
2. Maximizing resource utilization
3. Effective scheduling and eliminating wastage of resources.
4. Use of SCADA (Supervisory Control and Data Acquisition) boosts production efficiency by 6 percentages.

2.5 Software selection for using manufacturing shop-floor data in MPCS

This article [8] about MES confines role basically in Business process and Manufacturing Process, these two domains will determine the effectiveness and success rate of it. Software selection by far depends on following criteria:

1. Type of the industry that requires such system like Oil & gas, Food & beverages etc.
2. Global reach of the company in terms of production facilities.
3. Number of users associated with the manufacturing process.
4. Functionality required in the context of manufacturing data and interface.

MES implementation has to be carried with software and hardware installation in phase wise manner or in group. This has to include professionals from different sectors to carry out classified task along with active involvement of I.T vendor.

Different Vendors can be found in market to carry industry specific MES implementation that includes technologies from Rockwell Automation, Siemens, Invensys (Wonderware), SAP etc. Component of MES includes following factors:

1. Resource management.
2. Equipment maintenance/management.
3. Manufacturing Execution/Control.
4. Dynamic Routing.
5. Traceability.
6. MES master data management

Bottlenecks of Implementation of MES:

1. User resistance to changes in manufacturing and execution system
2. Time constraint for training of such huge system
3. Integration of legacy systems, with data source or communication protocol.

MES database system has to work in bidirectional, at one with business level data management system including SAP, Oracle, Microsoft database (SQL, MS Access etc.) and on the other hand with the shop floor database devices comprising of historian server, SQL, live servers. Shop floor data acquisition system is to gather production information from shop floor in order to support management planning, monitoring, and decision making. Barcode system are early method of tracking raw material , production data with rectangular bars with numeric or alphanumeric data associated with unique entity. Radio Frequency Identification (RFID) is a bidirectional communication with the RFID tag. On the one hand, it gets the stored data in the RFID tag and sends the received data to the further processing and on the other hand, the reader writes the data into the RFID tag.

The objective of this paper was to not find the perfect software vendor for MES application, but instead design a pathway to efficiently introduce the needs and importance of having such in order to increase production and sustainability.

2.7 BASIC INTEGRATION MODEL

A basic model with integration of ISA-88 and ISA-95 has been evaluated as shown in fig 2.5 with the understanding of the literature referred on ERP and MES with application of ISA-95 and ISA-88 respectively.

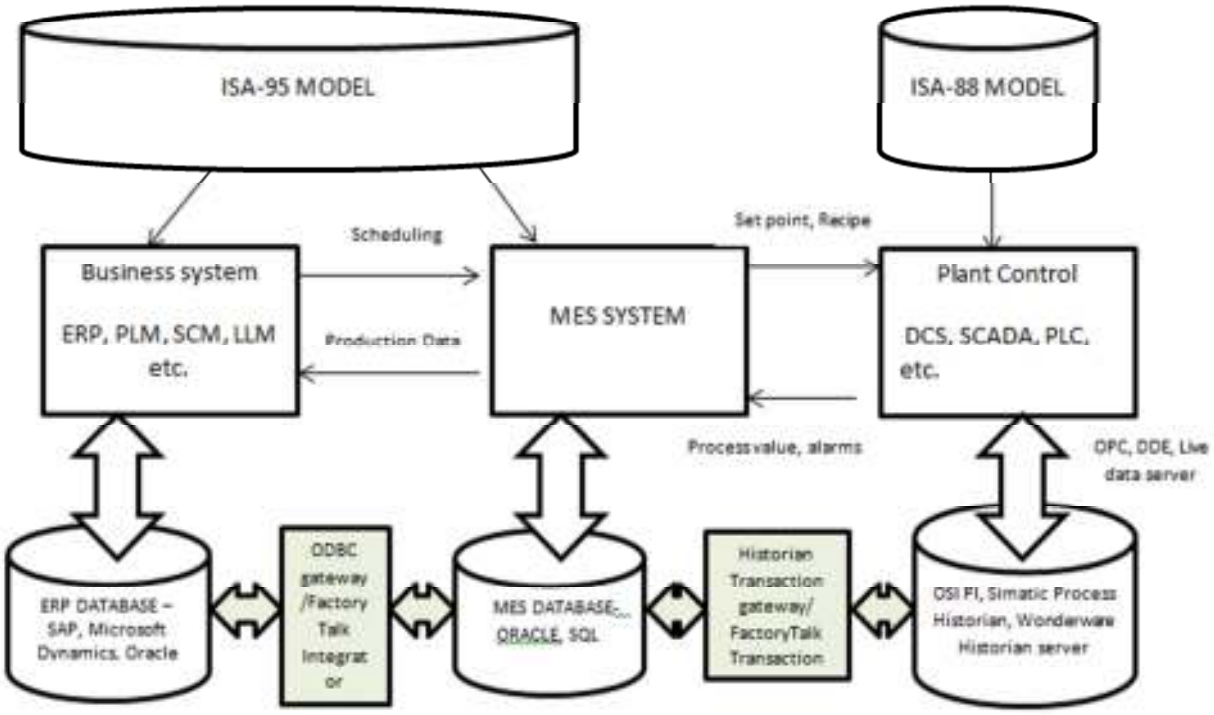

Fig 2.5 Illustrating integration model in context of manufacturing pyramid

3. Industry 4.0

A German government has promoted initiative known as "*Industrie* 4.0" projected towards smart manufacturing in future. Industry 4.0 talks about incorporating a paradigm shift in technological culture in manufacturing industries to about need of efficiency in production for future needs and also to ease the manufacturing companies develop fragility to reflect change in important policy making through the political decision making of the state. In short Industry 4.0 talks about industry to get ready for the needs of upcoming changes required in world in terms of not only technology as thing but also as efficient practice of that technology. Manufacturing industries faces a lot of backlash in terms of ever changing policy on global scenario to remain competitive and follow best ethical standards. [9]

A concept of smart factory where the devices from the production facility would be able to communicate with each other is desirable. Automation control system like PLC / DCS systems monitor and regulate these devices currently in Industry 3.0 that is advancement of electronics and Information Technology for automation. These two driving aspects are needed to be combined well together to form clusters of well intact technologies. Decision making is one the important scenario towards making smart or intelligent factories. Intelligent systems have four aspects i.e. sensing or feedback, learning, decision making and control. Today's automation has well advanced in terms of sensory control, and decision making up to a certain level of intelligence or knowledge extracted from the sensory data. Complex decision making calls for the application of "*Big data*" analytics and cloud computation where a huge network of data is available to every corner of the manufacturing unit that can take decisions without a human intervention. As a result manufacturing systems are integrated from the production layer to business process layer with horizontal integration of similar such production facilities.

Features of Industry 4.0: [9]

1. Interoperability: Efficient interaction between the human operators and computer driven machines also called as cyber physical system.
2. Virtualization: Virtual copy of the manufacturing unit helps in simulating the production facility through soft sensors that can assist in prediction and models.
3. Decentralization: Discrete decision making throughout the facility to reduce load on specific unit or Business entities.
4. Real Time Capability: Integration of Business process and MES will help in supplying important real time information to business process.
5. Service orientation :

6. Modularity: Ability to be flexible to bring out modifications in the future expansion or up gradation.

3.1 SMART Manufacturing with Internet of Things (IoT)

Integrating machines horizontally over the manufacturing process create a huge network that lead to enormous amount of data over the entire value chain. This data is extracted into information and eventually into knowledge from the Meta data. In the fig 3.1 showing an extraction of important related concept on every level, Data is the bulk of all facts and event occurred during and process.

With extraction of relationships among the keys in data, information is generated with such interlinking the keys from the ocean of data. With efficient extraction from information layer, Meta information is gathered which eventually helps in gaining knowledge of the process. The uppermost layer wisdom is the epitome of understanding of the knowledge of the process. This will be able to achieve through cyber physical systems, one of the important feature of Industry 4.0.

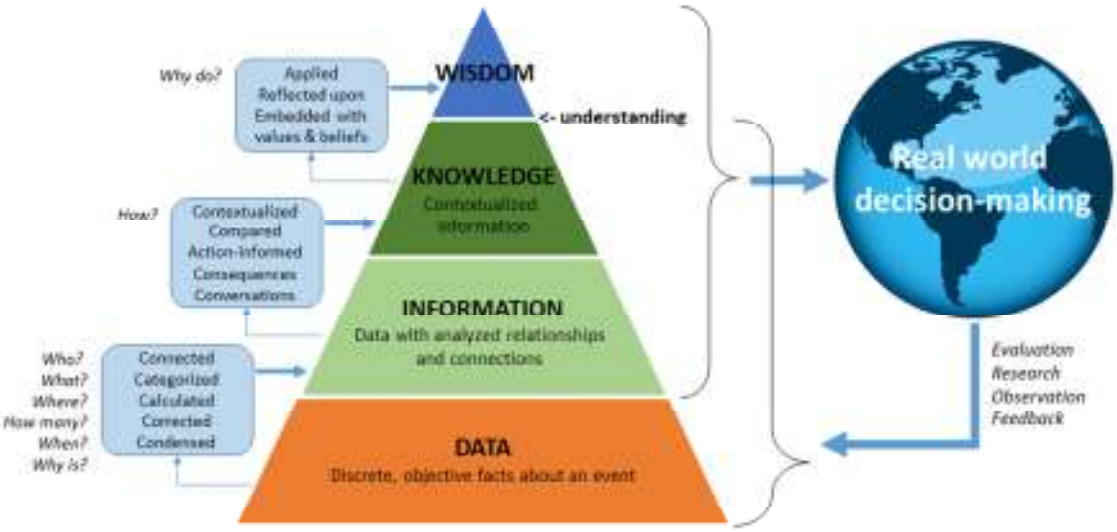

Fig 3.1 Illustrating Data Knowledge Pyramid [10]

IoT in the form of cloud computing and Big data analytics will help manufactures make smarter decisions and increase efficiencies in production. Concept of Smart manufacturing demands for the integration of productions units to business entities; such as MES, MRP (Manufacturing Resource Planning) integration with ERP, SCM (Supply chain management).

IoT has several possibilities of improvements within process in manufacturing that includes:

1. **Factory Visibility**: IoT and Industrial network forms cloud of network from the factory production floor affiliated to ISA 88 standard to ERP based systems and the partner chain network. A plant manager could be able to access the production from the sensors and actuator devices on a thin mobile device that will help him to evaluate proper action in response. The data cannot just be served to the production domain but also the ERP based business person through the web browser that can help him access over the World Wide Web. Rockwell automation has global visualization software tool called Factory Talk vantage point that helps in reducing downtime and visualizations for hierarchy of factory people from operators, supervisors and directors with differentiated visualizations templates available.

   Freedom in accessing the real time information and not just confined to a control room or a dedicated facilities for information sharing will put an ease in control and supervision of the facility. The more use of mobile technology will result in the reduction in time for the responsive action execution. The extension of visibility beyond the enterprise level, suppliers group and third party providers of services, consumers add to the decision making efficiency in the globalized market scenario. IoT will enable these global entities to be involved in the direct operation and maintenance with additional services that will create an entire new paradigm of relationships between manufacturers and suppliers.

2. **Automation**: Conventional control system is aggregation of PLC/PAC of various small systems that account for specific control and automation. On a larger scale these controllers are combined to form a network of distributed control system, but the distributed control system have a limitation of production unit location and cannot be extend beyond a certain distance. IoT with help of IP network generate a possibility of connecting these DCS systems located over distant places over the globe providing connectivity and information sharing over the partner business networks. As the sensors actuators and other machines are connected over through IoT, the huge cloud of information can be utilized to automate workflows for maintenance and optimize production with cyber physical systems.

3. **Energy Management**: Energy accounts to the huge share of consumable resources in any production facility, every industry has separate utility section to monitor and control power

system to the facility. Modern manufacturing demands for the investment efficiency in the power consumption. A lot of software's are available in this context to efficiently manage power and energy systems through modelling and management tools across the supply chain. IoT and automation can help in number of ways to reduce wastage and consumption in HVAC and electrical power. Interconnection of such energy management systems can help in adjusting the utility to be ready for peak demand through learning from big data. IoT enabled HVAC systems are capable of incorporating weather information and prediction tool to assist manufacturing industries in avoid unwanted expenses and plant energy need.

4. **Proactive Maintenance**: Preventative and conditional monitoring have been widely accepted but has been in progress from many manufacturers. Cost efficient sensors, wireless connectivity and visualizations on big data make its cost effective and efficient to retrieve real time data on monitor the equipment health. Manufacturer can be able to prevent equipment damages that has certain working operating conditions, for e.g. a sensor that can work in span of limited temperature range can be prevented from malfunctioning due to active monitor by a group of operators having access to the information of real time data. Similarly vibration parameter can be another example causing wear and tear of the equipment. Loss of such highly valued sensors and devices can put in huge cost deficiency for any business. OEE (overall equipment effectiveness), is one of the effective tool in reducing down time of the plant, eventually saving money by reducing production loss and allowing the company to undertake maintenance as per schedule.

5. **Connected Supply Chain**: Without a moment to spare assembling isn't another idea, however IoT, investigation and IP systems will help makers pick up a superior comprehension of the inventory network data that can be conveyed in real-time. By interfacing the generation line and adjust of plant gear to suppliers, all gatherings can comprehend interdependencies, the stream of materials, and assembling process durations. IoT empowered frameworks can be designed for area following, remote wellbeing checking of stock, and reporting of parts and items as they travel through the production network, among numerous different things. IoT frameworks can likewise gather and nourish conveyance data into an ERP framework; giving cutting-edge data to bookkeeping capacities for charging. Constant data access will help producers distinguish issues before they happen, bring down their stock expenses and possibly decrease capital necessities.

3.2 Implementation of Industry 4.0 for Manufacturing

We have seen IoT in manufacturing demands incorporating huge technological aspects and not a just a handful of them. Laying foundation for implementing IoT in manufacturing calls for at least four technologies described below.

3.2.1 Network

According to a cisco research only 4 percentages of all the devices on manufacturing world are connected to a network. [11] Industrial communication network has been a very prominent technological aspect in automation scenario. Communication is required on lower fieldbus network for smart devices that generates huge process data, such network are being like Profibus, ASI, Modbus etc.; also a layer above the sensor actuator i.e. supervision and control demands for efficient network for PLC, SCADA and DCS. A smart manufacturing calls for a master communication setup that drives bidirectional data over IP network in order to communicate with both production and business enterprise systems. An IP network also makes it easy to integrate and handshake with material suppliers and client customers in order to improve supply chain interoperability. Wireless network has distinct advantages in ease of operating alarms and transmission of real time data, but it does also introduce bottle necks especially in case of RF networks. Manufacturers need to setup a robust network in order to overcome this conventional constraint.

3.2.2 Security

Industry 4.0 talks of IoT application in smart manufacturing, with implementation of Information technology also brings the need to step the cyber security paradigm to prevent unwanted intervention of vital manufacturing data. Smart factories come with smart responsibilities and security. Plant managers need to ensure the safeguards are embedded in manufacturing solutions having freedom of IoT that's includes data security measures like hardware authentication for authorized users, security for physical locations and secured network access for data to be shared on the IoT platform. The manufacturing solution should be able to provide with a secured remote access to manufacturing systems. Security and network infrastructure also subjected to physical factors like harsh environmental conditions, heat and moisture that are unknown for conventional networks. Personal identification and authentication framework is required to be equipped to support "things" on network.

3.2.3 Cloud Software Architecture

IoT is about very different data that we normally use to work with. Thousands of sensors actuators giving out real time information, such a huge data requires a huge infrastructure or software

framework. This software architecture and framework should be able to extract information from the physical world into real time actuation and control that is implemented on plant floor by machines and humans. On a larger scale for the software for globalized enterprise scenario or interconnected supplier chain and partner companies the software should be secured, flexible and accessed from an established platform.

On demand cloud data exchange that facilitates the need to serve data among MES and ERP systems in context of ISA-95. Conventional IT application needs high capital investment and lacks flexibility of serving data on usage and on demand. Pay-as-you-go for manufacturing services and support during entire application cycle is highly important for the cause of reliable application framework. Radical improvements in terms of complex data structure and big data of process and lifecycle management in the application required for Industry 4.0. This will create a demand for a new ever improving software development that will be driving through huge capital investments.

Exercises inside the project incorporate characterizing ideas, making models and strategies for the assembling cloud as a complex arranged administration framework, building manufacturing specific information mining, procedure and streamlining techniques, prototyping chose examples of the reference design, and in addition creating situations and model demonstrators for assessment and acceptance of the proposed models.

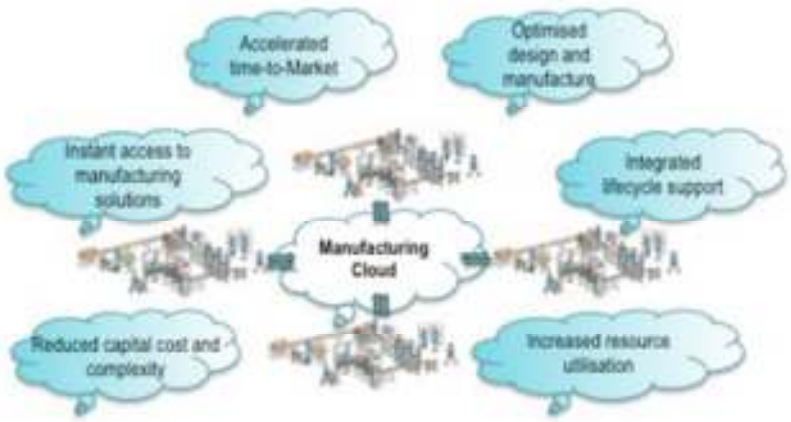

Fig 3.2 Manufacturing with cloud [9]

3.2.4 Big Data Analytics

While manufacturers have been generating big data for many years, companies have had limited ability to store, analyse and effectively use all the data that was available. New big data processing tools are enabling real-time data stream analysis that can provide dramatic improvements in real time problem solving and cost avoidance. Big data and analytics will be the foundation for areas such as forecasting, proactive maintenance and automation.

4. DATA EXCHANGE FRAMEWORK FOR MES & ERP INTEGRATION

4.1 Cloud application

Cloud computing is a broader term which is used in many application for many upcoming and new applications; it is more of an application deployment principle rather than a dedicated technology in the field of web application [12]. Since it is associated with wide variety of services and component in application framework, a lot of cognitive intelligence is required in deployment of such application. Such cloud computing services are commonly referred to as Software as Services (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Cloud application dedicates it role in serving I.T resources to various end users that can be termed as clients or tenants. A shared of pool of computing resources are enabled for remote clients which provisioned by a service provider without much of intervention and explicit efforts to serve the clients [1]. Some specific characteristic can be defined before implementing a cloud application for a specific scenario and that includes following:

- On demand self-service: Ability to access the application services only on sign in or specific time slot.
- Broad network access: Ease in accessing those services from a web interface through various available devices i.e. Laptops, mobiles, PDA etc.
- Resource pooling: A common source of services and infrastructure available for tenants.
- Rapid elasticity: Ability to modify and scale the services and resources for the tenant specific.
- Measured Services: Provision for the tenants to access only specific services needed that reduce a payload and cost in client context [12].

4.1.1 Infrastructure as a Service (IaaS)

It is a concept associated with providing computing resources in a virtualized environment [3]. In common terms the computing infrastructure is offered to a client who can virtually access the servers, network connections, and bandwidth. The client is made available resources to build its own IT application. Among the other cloud application types Saas and the Pass, IaaS is concerned more on resource virtualization and as infrastructure provider for the end user clients [3].

Salient Features of Iaas[13]:

- Enterprise Infrastructure:
- Cloud Hosting

- Virtual Data Centres

Benefits of IaaS [13]:

- Scalable resources is achieved which enables client to access and expand the limit of the usable infrastructure.
- Zero investment in hardware, due to the use of cloud based servers.
- Utility management as the client is made to pay only the resources it has been consuming.
- Infrastructure deployment is free from location constraint as the cloud service provider manages the centrality of framework deployment.
- Secured data centre through implementation from public and private cloud.
- Complete failure is avoidable through implementation of redundancy configurations between server and client.

4.1.2 Platform as a Service (PaaS)

In general terms it points to make available platform that serves necessary environment and services that will allow clients to build their own application over the internet [4]. Features that can be included in PaaS architecture are Operating System, Server side scripting environment, Database management System, Server software, Design tools and development etc. [4].

Software development, Web development and Business application are well suited to PaaS architecture. Paas have its own benefits which could be identified in terms as follows:

- Exemption from investing in infrastructure.
- Development ease for non-professional developers
- Flexible development environment with provision of picking up environment specific tools.
- Concurrency in development for multiple users to interact in a single framework environment.
- Data security and isolation for multiple user scenarios with recovery benefits.

In comparison to IaaS this deployment serves architecture and infrastructure both for the web application development.

4.1.3 Software as a Service (SaaS)

The most commonly known cloud architecture has been in the use for a long time that allows use of software environment over the internet. These applications were mostly built in single tenant

perspective unlike the modern applications in its initial days which had limitations over scalability and sharing of resources over number of client's users which in fact is not as economically benefiting unlike modern applications [14].

Today SaaS efficiently explores the use of Multi-Tenant architecture where centralized software implementation through single instance. Application is served by a service provider or aggregator that comprises of the multiple solution providers.

Benefits of Saas [14]

1. Centralized process for upgrade, uptime and security: A central framework running the application will handle the modification or upgrade for entire customer base with authorization protocol defined by the framework itself.
2. Reduced time to scalability and integration: Addition of costumer base on runtime is the key feature of SaaS. Incorporating services and functionalities is also handled in this type of cloud type.
3. Access software functionality from anywhere : A cloud deployment enables access to software functionality without any technical orientation, even a lay man is able to use the application with simple instructions.
4. Pay as per usage: Consumers are charged on basis of services used that are predefined by the service provider.

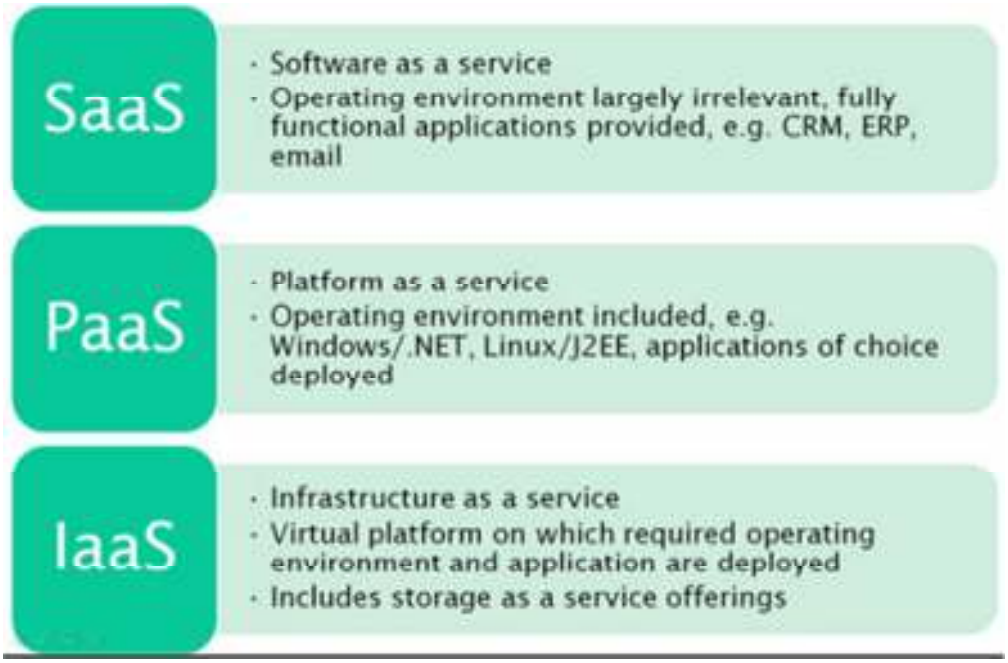

Fig 4.1 Illustrating in short detail of different cloud systems [14]

4.2 MULTITENANCY IN SAAS

SaaS is targeted towards audience which is a non-technical user, served with a ready to use application. The application also supports charging on basis of time consumed by the end user. So SaaS a very logical foundation for implementing a data exchange framework for a multiple company scenario in MES and ERP integration where customers are users inside the company itself.

Basic purpose of a web application is to cater its end user, other business entities and internal users of its own enterprises. Variety of web application is demanded in multi scenario case, for example a separate instance of web application can be served each user entity or tenant who hires the service form the cloud framework. Every client or user has its own version of web application in served which are called as Single Tenant application. [15] Contrary to this, data exchange makes a valid point to run a same instance with similar data structure to all the client entities, called as Multi-Tenancy scenario.

A tenant in the data exchange scenario is similar to conventional tenant renting a house. The house itself is SaaS model (application instance itself) which can accommodate single tenant or multiple tenants at a time. Single Tenant application is intended for multiple customers, but the instance is separate for all the users in the framework. While the multi-tenant application serves same instance for all the clients in the tenant model.

A multi-tenant application is a single codebase, installed on a centralized cloud server with single instance supporting all the required services for the application i.e. cache, database, servlets etc. A common strategy is implemented for all users in application, which relives from implying a different debugging for different users in case of error logging. All the tenants get updates and fix immediately after modified application is committed to the source repository. [15]

4.2.1 Architecture

Tenant is any entity either inside or outside the enterprise, which resides with other tenants without fear of breaching of data security from other tenants. Tenant entity itself is a multi-client, where a tenant object represents pool of users from the same tenant domain. Customization and data persistence is strict for all the users in same tenant model.

Apart from SaaS, multi tenancy can be applied to all the three cloud from i.e. IaaS and PaaS. There are basic features on which a multitenant application is pivoted upon; in short these specifications evaluate the efficiency in performance and functionalities. These features includes following

i. Costing structure: Service in multi tenancy can be charged on basis of tiered or flat costing.

ii. Resource pool: Multi-tenant application calls for shared pool of resources aggregated with services for all the tenant and clients within tenant. Client specific services are provided by the application instance.

iii. Framework maintenance: Maintaining the client list according to the role assigned in the application, administration of access rights and allotting relevant services is one of the important aspects in Multitenancy.

iv. Scalability: Ability to manage huge number of clients and their respective services, along with the dynamic addition of tenant models on the fly opens up new dimension in whole application scenario; in fact this application is very much in demand for consumer purchase sell applications.

In context of SaaS, it brings in more advantages in terms of additional features and described as

Data Isolation: In multi-tenant system a single database can be utilized for the entire application, multiple clients can be associated with a single database with separate schema or similar schema depending on the customization possible for every tenant.

Following table makes argument whether the single schema or separate schema is appreciated in this scenario.

FEATURES	Separate database	Separate schema	Same database/schema
Data customization	Highly supported	Possible	Not possible
Security	Offer highest security	Possible with authorization	Lowest security offered due to shared environment
Data inter dependency	Exclusive data size for every tenant	Exclusive schema similar to separate database	Prone to performance issue on data schema sharing
Scalability	Merely scalable as tenant grows data tables proportionally increases	Scalability possible as the schema can be expanded on the fly	Highly scalable as no expansion on database artefacts
Customer on boarding	Difficult due to lack of expansion on the	Moderately possible due to expansion on	Most appreciated in practice.

	fly	schema level	

Table 4.1 Detailed comparison of all Data level customization [16]

Feature Customization: In [16] dedicated environment there is a service which has a role only to serve a specific user in that environment. Unlike in multitenant environment a pool of services are associated with multiple clients or tenants. The mapping between services and clients are many to many, one to many or one to one depending on the scenario.

Customization includes in various following forms:

i. Data level: Customization on basis of persistence column, possibly addition or deletion on runtime.
ii. Functional level: Multitenant application is subjected to various users depending on the role assigned. For example an administrator in the tenant have the maximum access rights of data customization while the supervisor in the system could be assigned secondary level in the hierarchy and subsequently with the tenant user.
iii. Process level: Business process can vary from user to user, for e.g. one client may want material product integrate with context of Tenant entity A i.e. all the data will be related to a different tenant that consumes the material generated by the source client in tenant.
iv. Licencing specifications: The application requires customization on allotting licence depending upon the predefined strategy or cost subjective services.

Plug and play architecture plays an important breakthrough in achieving all the above discussed functionalities. [16] To see a development environment where all the modules relevant to the feature customization are added in runtime in the application flow, discrete software modules with a dynamic addition to the

Virtual Isolation: Data separation and Feature customization directs towards implying a virtually exclusive software environment. Environment should reside the entire tenant with a sense of data security that means every tenant should be able to allot an exclusive virtual partition on the required dimensions of data isolation and feature customization.

The prerequisites for Data Isolation and Feature customization suggest an execution domain confinement for every occupant. For e.g., in the event that we need to accomplish "Information Customization", it implies the POJO class that is utilized to load and spare the information should

be characterized diversely for various clients. This is impossible if the same execution environment is utilized for various occupants, since two variants of the same class can't be stacked into the same class loader. This additionally suggests the earth in which the code is sent can't be utilized as the execution environment for each of the occupants. Another execution environment must be made for each occupant and the right information source and code base must be empowered in the earth.

In java this can be achieved using Non-delegating Class Loaders. A new class loader can be created for each tenant and the licensed code loaded into the class loader allocated for a tenant. This gives us an opportunity to

- Deploy code without affecting other tenants in the environment
- Enable features for a single tenant without disturbing other tenants
- Remove functionality by just recreating the class loader for this tenant with the correct features enabled.

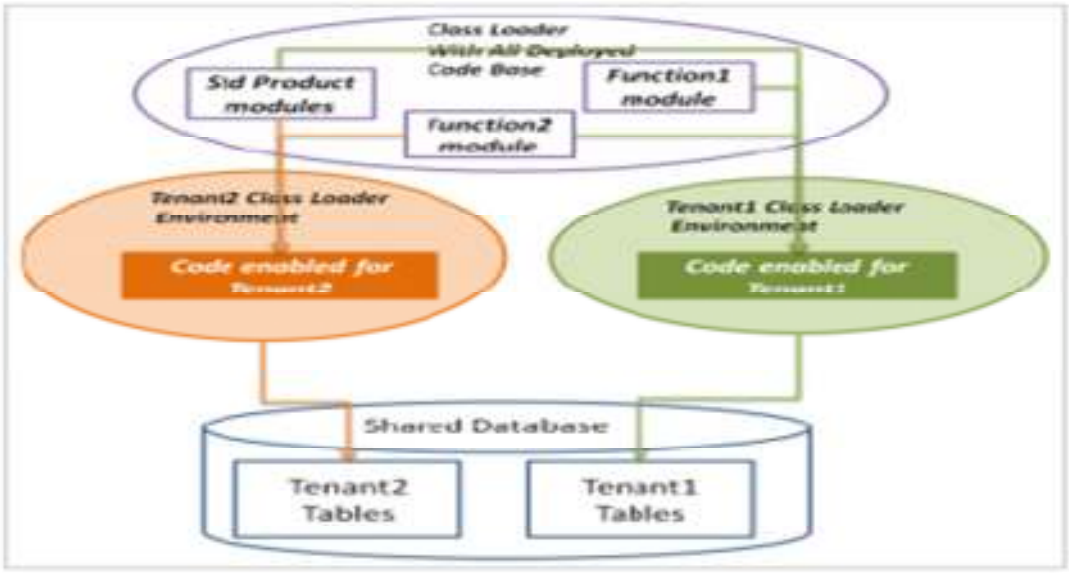

Fig 4.2 represents the aggregated concepts discussed in context to multi tenancy. [16]

4.2 Data Exchange Scenario

A data exchange scenario is considered where the material generated in MES system is fetched and extracted by the cloud application. Every product or material generated in MES during production is meant to be exposed on the Internet from where the data for the material is distributed among the tenant entities who are relevant to the material.

The multitenant framework is classified in two distinct paradigm, these are client and base framework. Clients are part of tenant entity, may be a group of identically authorized users or hierarchical clusters referred to as admins, supervisors etc. Base framework is core to the application of multitenancy visible to the entire application from the client and administration perspective. In detailed description of the entities in data exchange framework is mentioned as following and in fig 3.2

a) **Provider Entity:** A tenant entity in the framework, supply the list of the newly generated materials that are stored in the MES database. Provider generates a unique id for the material; in short the material fetched from the MES is literally translated in the context of multiple business entities that are nothing but clients in this case.
b) **Base Entity**: Constitutes the pivot for the Multi-tenant application, which is within the scope of the entire application visible to every client side server.
c) **Material Entities:** Also a client associated with specific tenant in the application. Basically these are the consumers within the manufacturing unit or users within the company or users from partner companies who receives the material relevant to them from provider through the base framework.

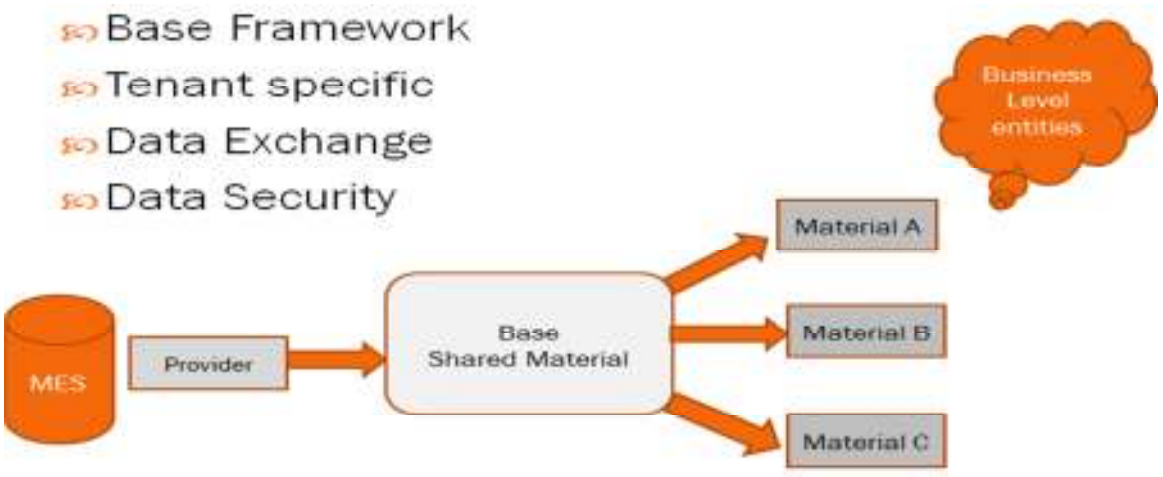

Fig 4.3 showing the data exchange scenario

4.3 Application Development

In context of industry 4.0, the data to be shared among all the user paradigms of cloud computing is through internet. A web application is a simplest form of realizing the need of sharing and storing the data from a remote web browser. Java EE is one of the commonly used platforms to develop web applications. It is an extended version of Java with strong support for web application development. A Java web application is accumulation of artefacts such as servlets, Java server pages, jars. Servlets is a Java class which handles HTTP requests from the client browser and returns the appropriate response from data model back to HTTP. Java server page is a mix of html and Java code, basically an interface to the Java environment. JSP and Servlets is core to develop the prescribed web application to be deployed on cloud platform. [17]

1.3.1 Grails Framework

Following are the features of Grails that are relevant to cloud web application development:

- A Groovy based web application framework that is based on the Java virtual machines (JVM). A smooth integration of groovy which is dynamic language based on the Java i.e. it compiles into the java byte code and Java itself is the advantage in commencing a web application development.
- It provides with the benefit of using huge Java background with already developed libraries and efficient and fast scripting like groovy.
- Groovy is a dynamic, language for JVM that is less verbose, relieving programmer of repetitive syntax just like a scripting language one like Python and Ruby.
- It compiles into Java byte code. Meta object programming, groovy is associated with MOP before executing on byte code on JVM making it dynamic on the fly compatible.
- Core to the framework is Java EE servlet API, which is foundational philosophy in development web application to grails. Also Grails incorporates the power of Spring framework, which a famous MVC framework based on JAVA EE giving an ease in developing a closed cyclic web application.
- Hibernate forms the persistence base with features of ORM (Object Relational Management).
- Grails also supports a unique feature of plugins which can be added to boost the functionality of an application through appending a plugin app.
- The Grails application developer community is highly active and ever improving due to the possibility of easy integration of plugins.

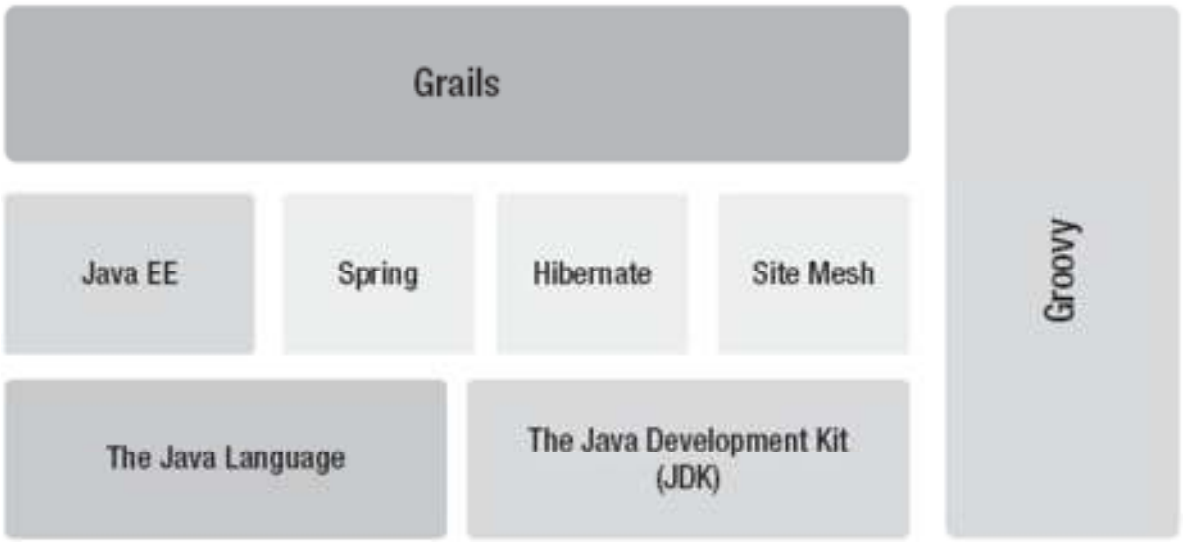

Fig 4.4 Illustrating the Grails Stack [18]

4.3.2 Class loading in J2EE context

Grails is a powerful web framework for the Java platform. Java class loading mechanism is utilized by the grails framework as well. Class loader is an object in memory of JVM that is associated with loading the Java Class on runtime.

JVM supports a default class loading mechanism that loads a class by predefined class loading objects during the start-up. To be able to build complex applications custom class loaders are well suited to be defined. JVM supports a delegation mechanism for class loaders to search for classes resources. Before loading of any class by a class loader object, it checks if the class has been already loaded by searching through the cache of the class loader. Every class loader is associated with a Parent class loader in JVM. Every child class loader or a custom class loader will always look through the parent class loader for loading the class if the class is not found in the cache of itself. If not loaded then it delegates the search to the parent class loader for the class or resource before it can search the class by itself. Even after a failure of parent class loader to load, the class is loaded by the child itself. A failure in all the cases will result to classnotfound exception by the class loader object.

As shown in fig 4.5 Bootstrap is the root of the class loading hierarchy in JVM. That means a Bootstrap will not delegate a search to any other class loader, it self-instantiated during an application start-up. It is subsequently followed by the system class loaders in the hierarchy as shown in the fig 4.6. A user defined class loader can be created with defining a specific parent to it but only at the bottom of the hierarchy.

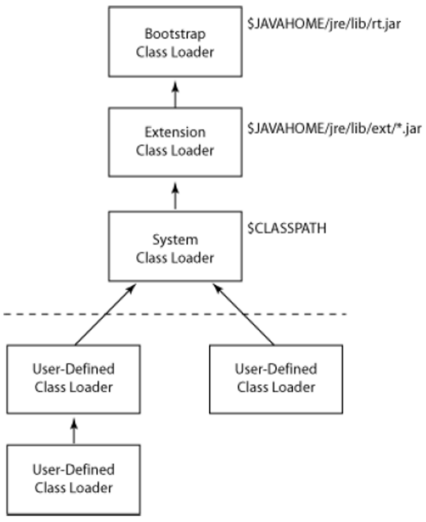

Fig 4.5 Represent a Class loading Hierarchy in JVM [19]

4.3.3 Class loading in context of Web application

Java Enterprise Edition (JAVA EE) is an extension of JAVA for web development. Java class loading mechanism forms the foundation for the JAVA EE but with an added perspective. Every J2EE application is packed as EAR (Enterprise Archive); every EAR has its own class loader. WAR (Web Archive) is a collection of distinct EAR, the class loading hierarchy is clearly visible in fig 4.6

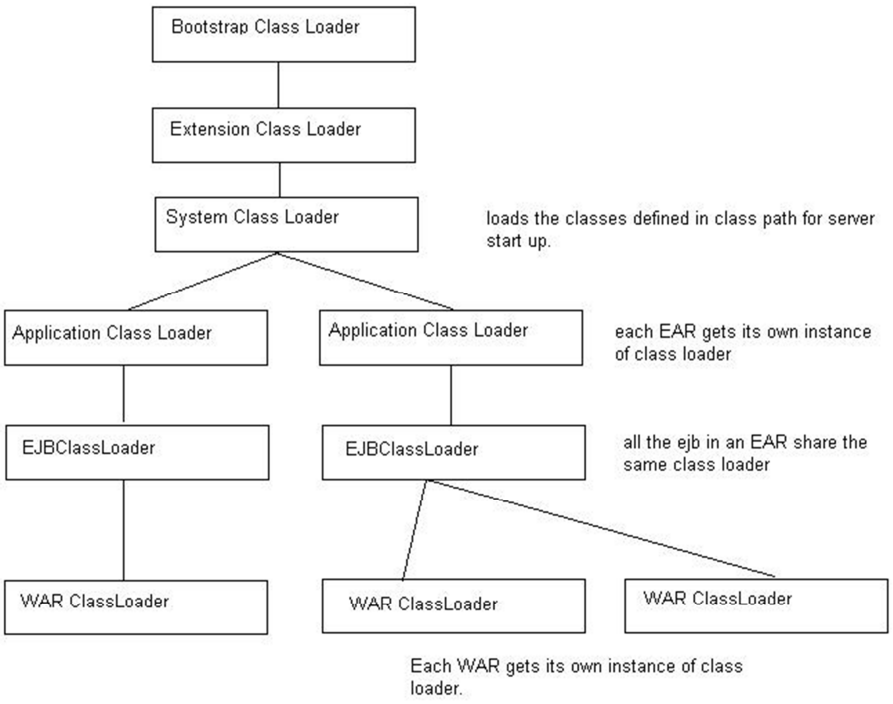

Fig 4.6 Represent a Class loading Hierarchy in J2EE [19]

4.3.4 Tenant Specific Class in Grails

In multitenant architecture, the tenant specific class demands for the data security especially with the other tenant models. For e.g. client in tenant Material A should be prohibited to access the other tenant classes Material B and Material C. Tenant specific class has a class definition which defines the schema of data model associated with it. Grails supports both Groovy and Java both which gave freedom in developing class definition as per convenience.

Tenant classes needed to be equipped with the ability to define definition from an interface that can be on HTML browser platform. Ability to customize the tenant class also comes with a constraint of virtual isolation already talked in the principles of SaaS Multitenancy. A tenant entity has to be secured in the grails framework application.

Grails application has its own class loader as previously described that every EAR application has its own class loader which is in down line hierarchy to the JVM class loading. The GrailsApplication API provides information about the artefacts inside the application, the structure of Domain class, Controller, Services etc. a detailed description will be described in upcoming sections.

POJO (Plain old Java object) is a normal java object that is not entitled to special role and no special interface associated with it. Every tenant entity is associated with its specific POJO in groovy also called POGO. To achieve data customization, every tenant has its own POGO class and has the ability to upload and save the new class definition.

Using a custom Groovy Class Loader to load tenant class on application start up from a specific URI in the server environment, groovy class loader loads groovy tenant classes can dynamically load the class on runtime of application.

```
def gcl = new GroovyClassLoader()

Class greetingClass = gcl.parseClass(new File (URI))
```

This will enables us to

- Deploy code without affecting other tenants in environment. The class loader of one tenant should not be able to load the other tenant, other tenant class are hidden to that class loader i.e. it is parallel to class loading parent-child relationship of other class loading mechanism
- With groovy class loading supporting reloading on the fly, the client can upload a updated definition with the child custom class loader associated with it to only the tenant specific class.

4.3.5 Multi-client tenant architecture

The application to be developed was meant to be designed in a perspective of sharing data among provider or supplier companies entity with the consumer client companies. Client in the data exchange framework is an enterprise entity will large number of internal users associated with it. Hierarchy of users have access to associated tenant architecture, tenant entity is group of users

Tenant is associated with a database model or schema in a database. The same schema is data source for all the users in the enterprise. Clients in the schema are not promoted to have their own unique data source either in terms of database or schema. This will only lead to complexity in operation maintenance of the application and large fragmented data sources.

Data persistence for every client is required; persistence is storing the instantiated objects from the respective tenant class. Every instance has to be stored in the database with a unique identifier that will be a key in performing CRUD (Create Read Update Delete) operations. Clients could be distinctly differentiated from each other on basis of access privileges provided during the runtime.

Role based security access, to simplify the tenant model by assigning specific rights depending on the hierarchical position of the user client. For e.g. Material "A" tenant could have its own administrator which is allotted to perform all the activities related to tenant specific task like manage the data source with all the access for CRUD operations. Normal user in the tenancy will be subjected to data read and write or a specific task which is not critical in enterprise context.

Fig 4.7 below shows the Tenant enterprise model managed by role allotting service by the application framework.

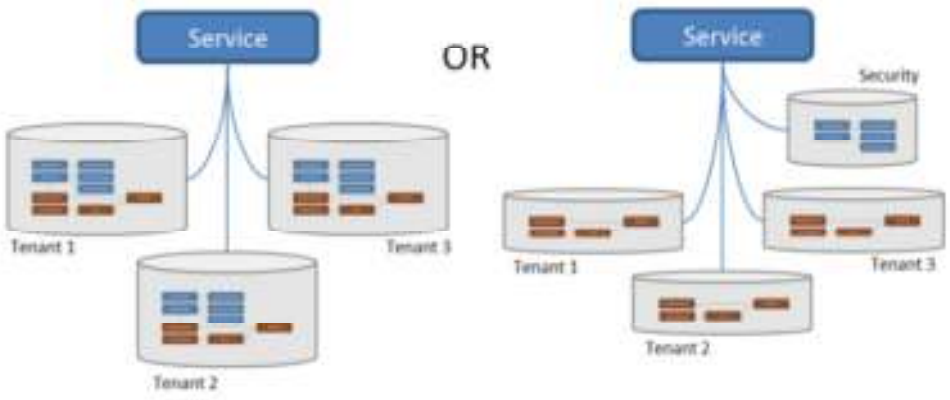

Fig 4.7 Service based model to handle tenant level operation [20]

4.3.6 BASE FRAMEWORK

Grails has a support for ORM (Object relational mapping), it is process of retrieving objects that are already persisted into the data source. GORM a grails version of ORM benefits from the dynamic language of groovy. Unlike other ORM, hibernate, Eclipse Link, GORM is based on convention over configuration i.e. doesn't required any XML mapping or added configuration for persistence and transactional behaviour. [18]

Grails uses Spring MVC to the core that's lays a foundation for the majority of application flow in the framework. Inside grails a domain class is under convention of grails-app/domain directory, all the groovy classes defined under this directory are domain class artefacts. By simple create-domain-class script, a new domain class is generated on application. Domain class inherits all the functionalities of GORM and core to any business application.

Base Material is a pivot object in the data exchange framework, which constitutes the dock for the sharing of dating between provider and client. The data relative to Base Material is visible to all the tenant and subsequent client in the framework.

Tenant specific classes i.e. Provider Class and Material client need to inherit the Base class to default access the attributes defined in the data sharing context. Base class as domain class in grails convention is a super class for all the tenant specific classes. Data exchange of persisted object from a provider source is served to all the other tenant class in the grails application framework.

Grails uses Spring's API under the hood also uses the Application context widely known among Spring developers. Spring IoC which is described later in section is used to load beans, wire them and make available during the application runtime. These beans are grails artefacts (Domain, controllers, services etc.). Domain classes are accessed or visible to entire application by spring bean factory or Application context.

Multiple Base class scenarios possible with ORM support by grails with Hibernate 3 under the hood stack with possibilities of relationship as following: [18]

i) One to One
ii) One to Many
iii) Many to Many

User Interface for Base class in grails provide with a built in script to generate appropriate views for the persistent objects. "Generate views" will create all the necessary GSP forms embedded with functionalities for CRUD operation. [18]

4.3.7 Controlling the application flow

Grails uses Spring's MVC framework that is architect around a DispatcherServlet that delegates the request to specific handlers. DispatcherServlet is a servlet inheriting HttpServlet class declared in web.xml and mostly prominent in J2EE. Spring MVC is a request driven controlled around a central servlet that delegates the respective controller to handle the incoming request.

The incoming URI request is handled by the DispatcherServlet, right set of controllers are evaluated for that purpose and views in of GSP (groovy server pages), analogous to JSP with groovy support. The following sequence event occurred during generating a view is visualized in fig 4.8 below

1. After a HTTP request, DispatcherServlet, consults to handler mapping to call the relevant controller to request received.
2. The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet. Controller holds business logic, that is defined for the respective incoming request and manipulate the domain (Model) class returning the view URL to the dispatcher servlet.
3. The DispatcherServlet will take assistance from ViewResolver to pick up the characterized view for the solicitation.
4. Once the view is finalized, the dispatcher servlet passes the domain model to the view that will be rendered on the front end browser.

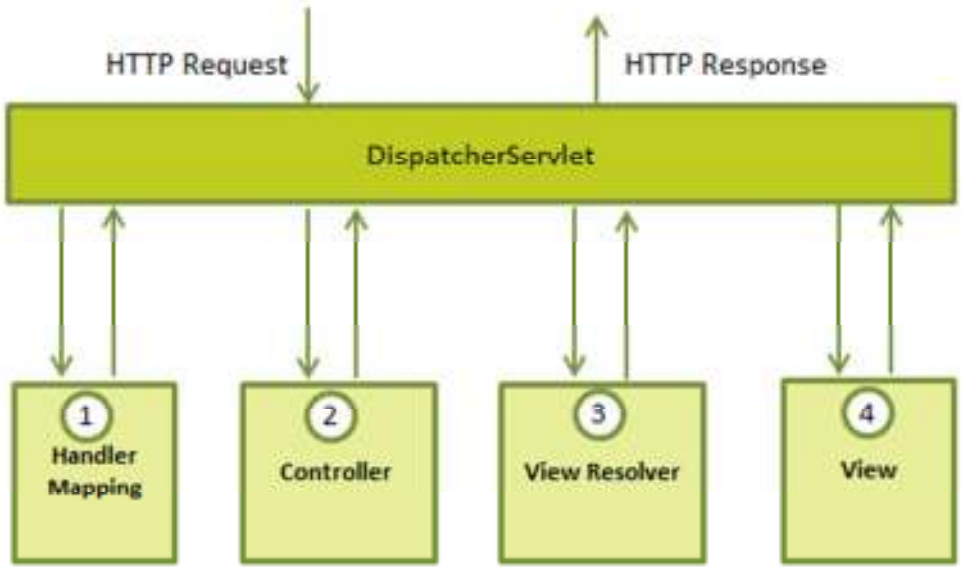

Fig4.8 Illustrates Dispatcher Servlet integration with internal application [16]

4.3.8 Accessing the Framework Business logic From Tenant specific classes

We saw domain classes are well equipped with all the functionalities in web application development in from creating a new definition, assigning a controller that incorporate methods to handle URI within the application and creating standard set of views in respect to CRUD operation. Creating a domain class fully that is equipped with user interface and database persistence fulfilled most of the requirement for a Base Material class.

To realize similar functionalities for Tenant specific classes is the next objective to perform. As discussed earlier they are not visible under the grails convention that means they do not have any special identification nor do they not enjoy the default data persistence and user interface under the grails frame. Tenant classes are simple POGO class. To start with development of Tenant sub architecture, it is important to understand about the Spring IoC (Inversion of Control).

Spring IOC also known as Dependency Injection, is a process of passing the objects required by dependent resources or class. Normally a class depending upon some object of other class calls the external resource by itself and the dependency is active only when the dependent object instantiate or reference it. But the contrary method where an external container handles the dependency objects is known as IoC.

Grails uses Spring IoC to the core in the application flow. In spring, the items that frame the foundation of your application and that are overseen by the Spring IoC holder are called beans. A bean is an item that is instantiated, collected, and generally overseen by a Spring IoC compartment. Something else, a bean is basically one of numerous items in your application. Beans, and the conditions among them, are reflected in the arrangement metadata utilized by a holder. As clearly seen in fig 4.9

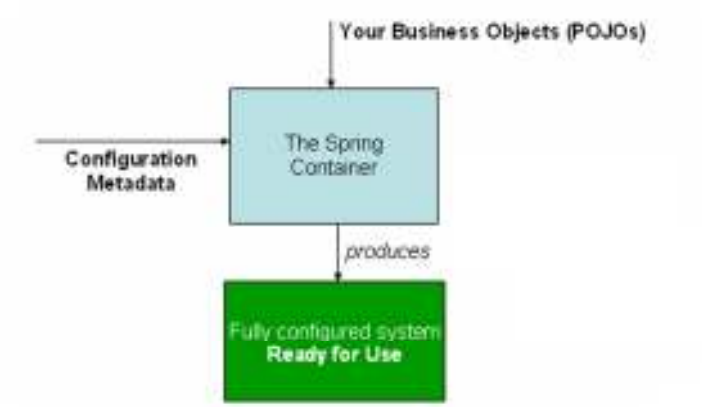

Fig 4.9 Shows Spring IoC interaction [21]

Several artefacts that are created in grails application are created as spring beans and specially services which allows the most control. The spring container core to spring framework in grails is managing these beans throughout the lifecycle from creation of that bean to its destruction. Every bean is visible from entire application scope i.e. controller can inject services by simply defining the service name in the controller context.

In the fig. below, services as singletons (instantiated once during start up) are embed in the controller action; whenever a controller method is invoked the method invokes a method form service class. Services offer a simple, maintainable, and testable way of encapsulating reusable business logic. Services can participate in transactions, be injected almost anywhere in your application, and are easy to develop. It's time to abstract your Post operations into a PostService that you can access from anywhere in your application. Controller is free most of complexions due to lot of code inside making it dedicated to redirect and interactions with HTTP request and response.

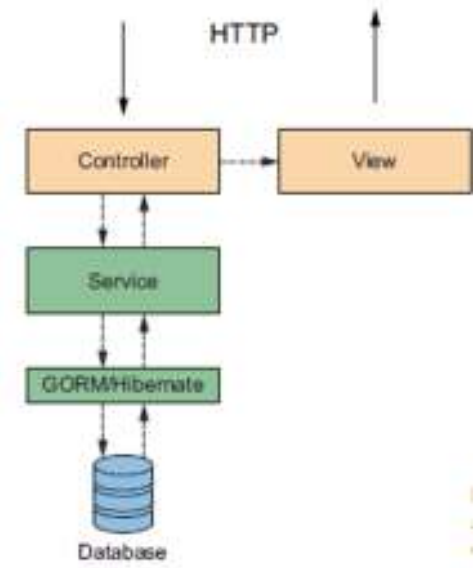

Fig 4.10 shows use of service in encapsulating business logic. [18]

Base Material class as a Domain artefact is handled well by the grails framework; domain is bean on application context which can be injected into other grails artefacts like controller, services etc. and also wire together with other bean objects with Dependency Injection (DI). Data persistence is easy to implement for the Base class with transactions and CRUD default methods.

In fig 4.11 shows bellows illustrates the wiring together of controller, service classes by the Spring IoC container used by Grails.

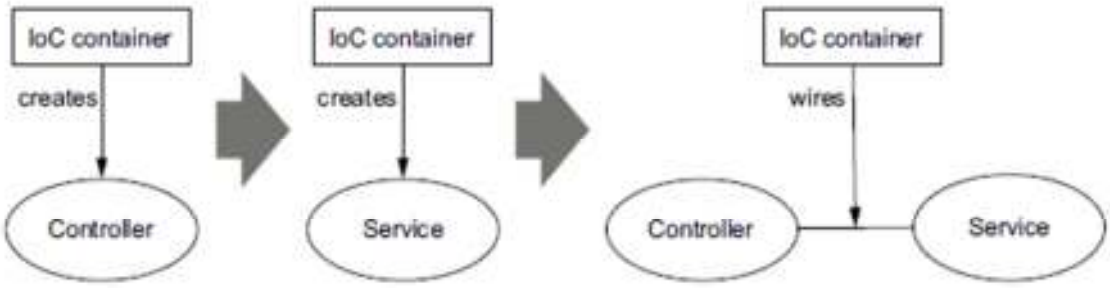

Fig 4.11 Shows IoC in Grails application [21]

With transactions a series of methods related to database actions are aggregated into single unit. Transactions make sure the database related operations are either executed completely or aborted in case of partial execution. Transaction management is a critical part of and RDBMS oriented web applications to make sure data integrity and consistent information. Concept of transactions has following properties: [19]

1. Atomicity : Single unit operation for data persistence
2. Consistency: This represents the consistency of the referential integrity of the database, unique primary keys in tables etc.
3. Isolation: There may be many transactions processing with the same data set at the same time, each transaction should be isolated from others to prevent data corruption.
4. Durability: Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

4.3.9 Service handling of Tenant Specific classes

Tenant specific classes are hidden under the grails convention. If the tenant specific classes are setup as domain objects, data persistence and transactions would be effectively placed. To expose the tenant class on grails application disqualify the objective of data isolation among different tenant clients. Virtual execution environment is one of the important factors in developing multitenant application.

Remember the Parent-Child class loading mechanism in Java. Child class loader is able to access the class loaded by its associated Parent loader. Tenant class as POGO class loaded by custom class loader, able to access the grails artefacts especially service classes that handles the request for the URI handler. Service singleton could invoke method implementing the insert, read update for the tenant entity using the Java API for relevant database.

4.3.10 No SQL database Solution

NoSQL has becoming increasingly popular among application of big Data with huge data volumes and variety. Relational database (RDBMS) is simple and easy to work with good support n tools for monitoring and backup. RDBMS does not possess the flexibility, agility, scalability required for the multitenancy architecture. Data customization is an important aspect in tenant specific where tenant admin has the access to modify the class definition.

RDBMS server process reflects single point of failure; the entire database server will be shut down in case of abrupt crash or up gradation or modification. The multitenant is a distributed architecture with multiple enterprises having stake in it. In case of failure at central database location, entire data source stakes are risk bring down the entire business operations of all the suppliers and material clients associated with it.

RDBMS are based on schema model which constitutes for its consistency and performance. Tenant customization requires frequent modification of data definition which in turn calls for a refactoring the schema in database.

Multitenant application is a scalable architecture, grows horizontally as the tenant model are expanded on the fly during runtime. Single node RDBM will never serve the functionality of horizontal expansion with limited resources in server like RAM, CPU, storage disk etc.

4.3.10.1 MongoDB NoSQL Solution

NoSQL MongoDB provides following features over the shortcomings of RDMS in multitenancy:

1. Flexible Data Model [22]: NoSQL effectively overcome the issue of relational database dynamic schema. MongoDB supports a document, graph, key-value, flexible data model with possibility of dynamic change in schema without shutdown of database server.

2. Elastic Scalability [22]: NoSQL database are built on focus with horizontal scalability with a built in relationship support. Application stores objects in a schema free structure, non-hierarchical structure to incorporate large files of data blocks.

3. High Performance [22]: NoSQL database for instance MongoDB are developed to provide efficient performance, in case of both the performance characteristic of throughput and response time to query.

4.3.11 GORM MONGO DB PLUGIN

With Plugins in Grails it is possible to extend functionality of the application just like appending a project which is been previously created to the existing one. The plugins can be downloaded from the grails repository.

MongoDB is a balance between key-value stores known for latency and scalability, and conventional RDBMS providing rich queries and specific functionalities. Features supported by GORM MongoDB Plugin mentioned following GORM [23]:

1. Simple persistence: MongoDB plugin allows GORM (domain class) to CRUD basic operations with database schema.
2. Dynamic finders: A dynamic finder resembles a static method conjuring, yet the strategies themselves don't really exist in any structure at the code level. Rather, a technique is auto-mystically produced utilizing code blend at runtime, in light of the properties of a given class.
3. Named Queries: Similar to *Hibernate* named query, where key finder is any name, just like alias name. It facilitates the programmer by avoiding writing complex queries in the code.
4. Inheritance: Inheritance mapping possible with table per hierarchy, table per concrete class, table per superclass.
5. Embedded Types
6. Query by example

Using the MongoDB Java driver directly makes possible access to lower level API. Tenant specific classes are possible to persist to MongoDB data source with the Java driver directly. Tenant specific classes can be persisted getting hold of mongo instance (com.mongodb.Mongo), implemented inside the tenant specific service. Remember as previously explained Grails Service are possible to be injected in POGO Tenant classes through Spring IoC integration.

4.3.12 Authentication for multi-client Tenancy using Spring Security core plugin

Spring Security core plugin embeds Spring security into grails application. This plugin provides an easy, simplified implementation in applying authentication and security to grails application. Spring security in itself is a powerful customizable authentication and access control framework. [22]

Spring security provides aggressive security to the grails application; on request to the DispatcherServlet URI responded by the controller is initially blocked unless there is authentication matching or the logged in user matches the annotations on the controller method that redirects to the requested URI. That means the URI is blocked unless there is request mapping even if the rule allows the access. [24]

Plugin uses @Secured annotation to secure the controller actions which are methods or closures. Default authorization is used by plugin to configure specific roles to specific action on the controller.

```
@Secured(value=["hasRole('ROLE_ADMIN')"], httpMethod='POST')
def someMethod() { ... }
```

Role based hierarchy has been an important point in enabling a tenant model which a domain with sub clients differentiated according to the access rights i.e. admin, supervisor, normal user. Role hierarchy class generated by Spring security core plugin allows explicit configuration for the setting up the hierarchy in grails application. This facilitates the user with highest priority for e.g. admin to inherit all the access rights allotted to the lower priority authenticated users. This relieves the programmer of setting up additional security implementation explicitly.

Spring security incorporate following features in grails application: [25]

1. Authorization and authentication supported both by the plugin
2. Integration with servlet API, easy to implement in controller servlet in grails simplifying the controller action secured access control.
3. Protection against malicious attacks like [25] session fixation (hacking a valid user session), clickjacking (injecting malicious links under legitimate links), cross site request forgery (overriding unknown commands for client).

4.3.13 MES integration with Quartz Plugin

The main purpose of the multitenant application development is to create an integration scenario between material supplier integrator and partner consumer client. The source of material data which supplier provides is shared across the cloud platform through a reliable Base material which is core to the application. MES is hub of all the material especially the finished goods within the supplier enterprise. The other end of application is multi enterprise ERP systems.

Provider tenant class holds the task to integrate with its MES system, data source in specific. The grails framework needs to control or invoke method from the tenant specific classes from provider tenant. The data objects reflecting material details from MES are supposed to be updated in the Base Material with the list of data items attributes defined by the provider tenant class definition.

Provider tenant class definition requires a data hook, a method or a closure defined under the tenant class enlist the parameters to be retrieved from MES external data source. A periodic or event triggered method to successfully poll data from MES database can be adopted. In this case a periodic polling of data from external database is achieved using Quartz plugin in grails.

Tenant class encloses a public static method that can be executed from the Grails framework after the tenant class loaded by the custom class loader. The tenant class is able to execute a service class using grails servlet context which is similar to application context but on Web container level. The service class calls a Job artefact created using Quartz plugin.

Quartz plugin [26] allows grails application to schedule task by a periodic interval or cron expression (string representing time-date) to execute them. An actual implementation of data hook between tenant classes and Grails Base framework through "*class TestJob"* using a periodic trigger that executes a *public static interface* from a hidden tenant class every 5 seconds.

```
class TestJob {
        def providerService
  static triggers = {
  simple name: 'mySimpleTrigger', startDelay: 5000, repeatInterval: 10000
 }
 def group = "MyGroup"
 def description = "Example job with Simple Trigger"

 def execute(){
  print "Job run!"
        //println providerService.providergbvar
        if(providerService.providergbvar){
        Scheduled.updateables.every { entry ->
                entry.key."$entry.value"()
                println Scheduled.toupdate

                }
```

4.4 Results & Discussions

The research study was conducted on the Grails platform to evaluate the possibility of developing multitenant SaaS architecture with core principles of following:

1. Data sharing and Exchange.
2. Integration with external environment.
3. Tenant security model.
4. Data persistence of Base and tenant classes.
5. Tenant security.

This multi tenancy proposed in scenario of data exchange framework in context of Industry 4.0 to serve data received from MES system on a global enterprise level through internet. This enterprise is partner clients that share material generated by Supplier enterprise. Enterprise or tenants visualized as tenants in the multitenant cloud application. The enterprise application was developed to figure out the possibilities of creating such architecture and not to create a ready to deploy cloud application. All the objectives were tested in a development mode of Groovy/Grails Tool Suite on grails version 2.5.4. An integration model in this context is illustrated in the fig 4.12.

Single Instance shared over internet: Grails is shipped with a tomcat plugin used to host application during development process. Quick hosting of application by grails on server with application name in suffix to server, application can be run in browser will URL as "http://localhost:8080/multitenant_app". A level 2 of SaaS model was realized with this implementation; single instance serving all the users in the application reduces the load due to excess strain on resources of server like memory, processor, RAM (Random access memory) etc.

Tenant data security: Unlike the conventional multi-tenant applications, where the data security is more or less limited to secured data source access on tenant level, this application follows a different path i.e. preventing CRUD operations for other tenants through class Java loading mechanism. Secured class loading mechanism is implemented for every tenant using a separate class loader. Custom class loader hidden in grails convention model, an object of the loaded class only accessed to the logged in user.

Data Sharing through Base framework: Spring integration in grails enables use of Application Context which serves parent class loaders to child tenant specific class loaders. Services are easily injected and accessed simply by defining an object in other classes, even by the tenant classes.

Tenant customization: Update of class definition possible due to reloading of Groovy class loaders. Persistence for tenant class's does not needs major refactoring compared to SQL persistence, MongoDB proved to be flexible enough for simple POGO.

Secured authentication: Grails plugin really promotes faster application development with just complimenting the existing application with added features developed in other plugin projects. As in case of Spring Security core plugin, only added explicit configuration was role based "Secured"

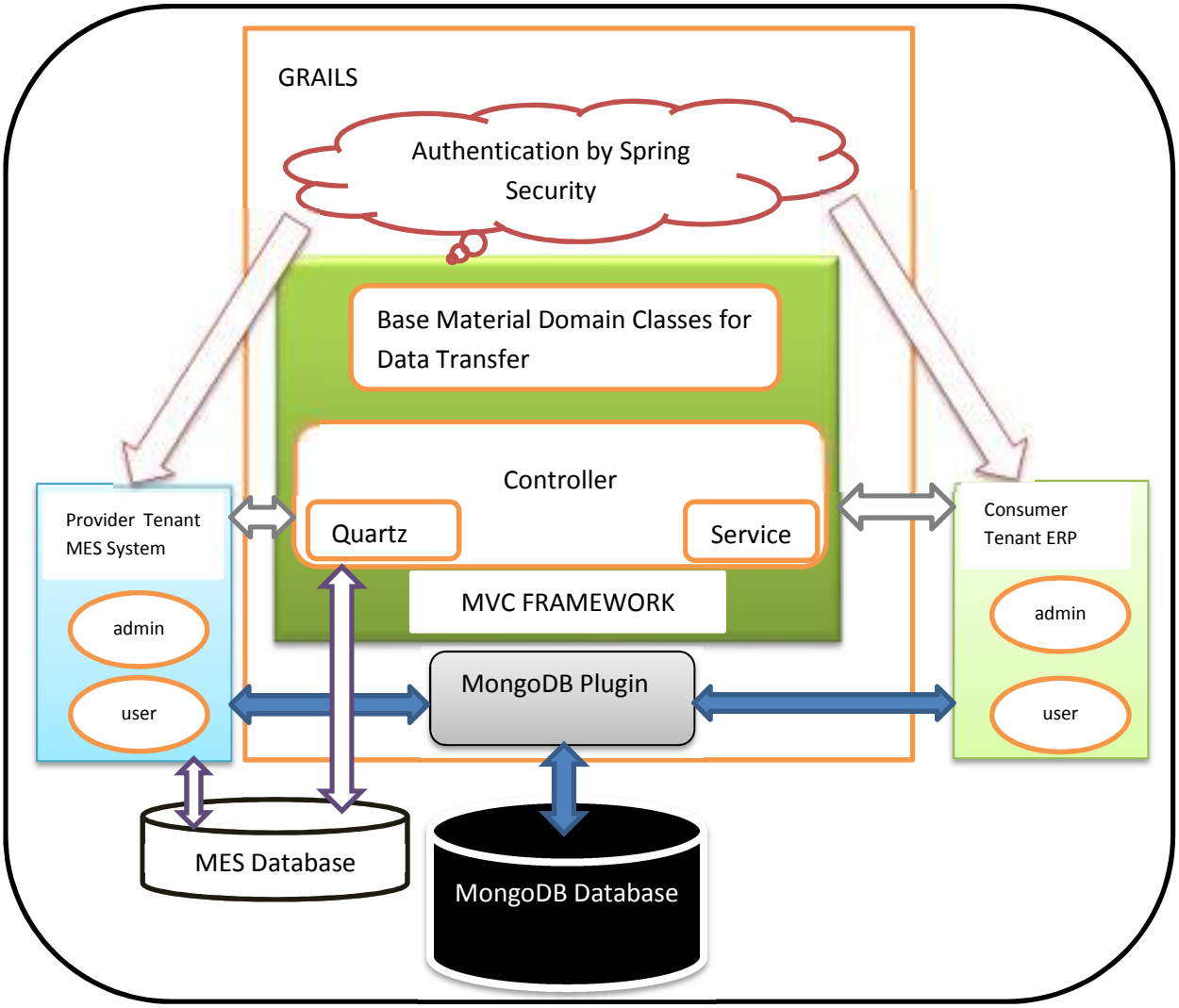

Fig 4.12 Represents connected technologies in multi-tenancy using Grails

Limitations in current development and Future work

1. Customizable Front end for each client in Tenant model.
2. Scalable client entity, ability to dynamically add users during the runtime process with least possible explicit configurations required.
3. The result of this study serves only as a foundation of understanding how a multitenant system can be implemented in grails, which is a tool for gaining wider insight into multitenancy application of integration MES and ERP.
4. A lot of efforts require to generalize this design on cloud platform

5. BIG DATA AND VISUALIZATIONS

Digitalization in manufacturing already has a huge stake as key for monitoring, control and performance evaluation. Digital factories amount to huge data from sensors and other devices, data generated from cyber physical systems. To efficiently use this data information has to be extracted from the huge amount of data. With the help of analytics and visualizations tool the data will be useful in key decision making in manufacturing production. We have simply research on developing visualization for an electric grid using modern HTML based open source tools.

This methodology on visualizations presents the result of an application of graph theory on topology visualization and identification of electrical grid. Electrical grid is connected network comprising of electrical equipment to make electrical power available to the distributed loads or consumption devices that are placed over specific locations.

The objectives of the experiment were to apply principles of graph in representing the electrical grid network. Electrical grid gives an overview of all the physical connections of buses, breakers and loads, basically these are the elements which are to be considered while applying the principles.

The grid is considered as supergraph of a connected subgraph which is important aspect in identifying the connected topologies. The fragmented topologies are the status of the grid for a particular time instance. The idea to work on represent electrical grid with a simplified structure was considered in a context to avoid complexity in viewing the standard grid. Electrical Grid is a collection of multiple electrical devices that have a physical connection across them routing the electrical power transfer from generation point to consumption. This made quite interesting to start working on a representation that will efficiently describe the physical connections as well as reduce the complexity of those connections.

Grid is aggregation of multiple small networks or fragmented networks depending on the active status of the electrical components. Displaying complete grid may have fall short of the objective in ease of monitoring the grid. The island of small networks within grid was given equal attention as in case of the complete grid representation. These smaller networks would be seen as instrumental in studying the connected topologies of the grid. The reduced representation may facilitate the ease in monitoring with relation based between the power generation and consumption devices across the grid.

5.1 Graph visualization using D3

Basically a JavaScript library with inclusive of HTML, CSS features to bring out variety of visual attributes with data driven approach to the SYSLAB grid data. Reusability of code was the main attraction towards adopting this library and which also helped in ease of development.

D3 had a variety of applications on representing data on different layouts like Force, Tree, Hierarchical, Packs etc. Force Layout was selected in this application and most suitable to application of undirected graph visualization. A two dimensional representation of the nodes and edges with all the nodes exerting negative charge on each other , eventually forming a spread out layout . This spread out format leads to reduce in congestion of edges, thus forming a less overlapped graph for the SYSLAB grid network.

Given a practical approach towards identifying Topology for the SYSLAB grid, the solution was intuitively pointed towards the application for the Graph. The elements in the SYSLAB network are nothing but the breakers, Buses, and loads. These elements have physical linkages in an actual grid i.e. Buses- Breakers or Breakers- Lines are possible combination for the visual representation for complete grid.

5.2 JavaScript as development language

A simple scripting language which is easily supported on client end was the first idea to go with this development. JavaScript supports numerous graphics Library which was the key factor in initiating with the application development. Also its simple front end incorporation with HTML-5 based web pages that supports almost most of cross platform OS.

5.3 Static Topology Visualization:

Static grid View was assumed as exact replica of the electrical topologies that constitutes the SYSLAB grid. The visualization was simple to figure out from the SYSLAB architecture. D3.js supported a number of layouts and as well as custom layouts that gave a freedom in creating a custom SVG representation

The static grid visualization was commenced with redrawing a sample electrical topology as shown in Fig 5.1 D3 did allow variety of method to be implemented using various SVG and CSS style, but the development was quite

Instead a layout already existing from the D3 recommended templates was adopted. There was need to restructure the elements for that layout in terms on Breakers, Buses and Loads.

To initiate with pragmatic application it was necessary to have sample electrical topology which represented as building block for the actual SYSLAB network grid, which is illustrated in Fig.5.1

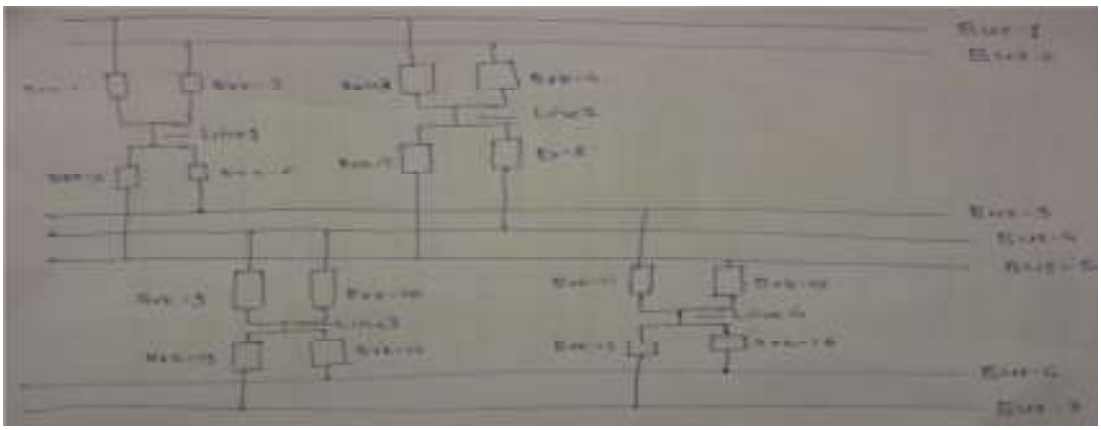

Fig.5.1 Test electrical topology for static visualization

5.4 Implementation in D3 Library

a. Data collection

The development of application had to be initiated with extracting the breaker status information from the available dump file. These breaker statuses were modified to identify the status of respective loads and lines connected to it. All the breaker status and abstract status of buses and loads were to be properly place in an array of input data that will feed the visualization input for the subsequent graph visualization.

Read CSV dump file using d3.text API: Reads the SYSLAB dump file in CSV format and returns the string in form of array of text. The array of text simplified the operations on specific strings with just mentioning the row and column of the specific part in whole text. It also made easier to extract respective data for particular header. As the SYSLAB dump files contained information of breakers, and other electrical specifications current, power, voltage etc. out of which only relevant information required was breakers.

Timestamp recognition: The data required for the complete topology was not comprised only in the single document instead it was dispersed over the multiple files. Also the timestamp identification from each file was never to be exactly the same as the argument specified in the Input Box may in case not exactly the one present in file. Closest Timestamp recognition: Formation of an array of Timestamp prior to find the closest value holds the key in this algorithm.

```
For (i in array) { var m = Math.abs(num-array[i]);
if(m<minDiff){
minDiff=m;
```

```
ans=array[i];}}
```

All the files are read asynchronously with every application of D3.Text function. The files depending on the size take certain amount of time to be read and stored in form of array of text. On the reading of final file, an aggregated array is computed which contains breaker status from all the section of SYSLAB grid network for the closest Timestamp.

Determine status of bus, breaker and load: Buses are always active element in the graph, i.e. they are the power source for the network or subgraph. Lines are specifically active depending on the breaker which connects them to the Bus and similarly does the status of loads. A final array of active elements is computed with only the unique no. of the respective node specified

b. Static Topology Visualization

A standard representation for all the nodes and links present in the SYSLAB architecture are considered in this case. This is in fact a static layout with all the static position of breakers, buses, lines and loads. The data for all the nodes and Links are sourced from a static file which holds a JSON object of Nodes and Links respectively.

```
 {"nodes": [
  {"name":"Bus-1-1172","group":0,"type":1,"pos":10,"x":40,"y":140,"fixed": true},
  {"name":"Bus-2-1172","group":1,"type":1,"pos":50,"x":440,"y":140,"fixed": true}

],
 "Links": [
  {"name":"Link1","source":2,"target":3,"width":20},
  {"name":"Link2","source":2,"target":4,"width":20}]}
```

Node Element with following attributes:

1. **Name**: Static name for every object that matches the name in the Dump files
2. **Group**: Unique number for every node i.e. also an identifier for a node
3. **Type**: Breakers (-2), Buses (-1), Lines (-3), Loads (-4) category numbers for colour identification.
4. **X**: static x coordinate; **Y**: static y coordinates for the node.
5. **Fixed**: Boolean property whether the node is clamped or not.

Link Element with following attributes:

1. **Source:** Group no for source Node.
2. **Target**: Group no. for target node.

A directed force layout was applied to this method with the JSON object holding the key data for all the respective nodes and the links connecting between the nodes. Since as per the method Undirected graph satisfied the objective of the application, directed information of links i.e. in terms of Source and Target accounted as mere Union of nodes rather than direction of link.

Force directed Layout: It is a graph drawing algorithm that places a nodes and edges in two dimensional planes. D3 Force layout uses a quad tree approach towards drawing graph. Application of quad tree comprises recursive placement of nodes and edges using Barnes-Hut approximation. Barnes-Hut simulation is approximation algorithm performed on n number of elements having order O ($n \log n$).

Algorithm:

1. Algorithm Force(Graph);
2. Place vertices of G in random locations;
3. Repeat M times calculate the force on each vertex;
4. Move the vertex (force on vertex) draw graph.

c. Connected Component Algorithm

The heart of the application i.e. Subgraph enumeration is achieved using the application of Connected Component Algorithm. This Algorithm significantly distinguished into two sections:

1. Searching the connected Tree or spanning tree.
2. Identifying each tree or subgraph with unique serial no.

Data Structure for Search Algorithm: Unlike the data structure which was in form of list of objects to determine the nodes and links in the graph. One similar to it was not suitable to apply a search algorithm i.e. Breadth First Search or Depth First Search.

A reduced version of the Adjacency matrix Data structure, it forms an efficient representation of the subgraph and at the same can be subjected to DFS search algorithm recursively.

E.g. Table 5.1 shows adjacency Matrix [27]

Vertex No.	1	2	3	4
1	1	1	1	1
2	1	0	0	0
3	0	1	0	1
4	0	1	1	0

Adjacency List for corresponding Matrix in Table 5.1:

```
1:2, 3, 4

2:1

3:2, 4

4:2, 3
```

Adjacency List needs to be evaluated for the list of all the active nodes. For an Undirected graph representation the Adjacency List for any Vertex I should contains Vertex J connected to it with and edge and at the same time Vertex J will have Vertex I in adjacency List of it. Input Data array which contains the active node elements will be the feeder input to the algorithm, from the offline status of the links between all the static nodes. From the Adjacency list mentioned above, to implement the same in the application needed algorithmic operation on every active node. Every node has adjacency list element in data structure that was filled with the connected nodes on increasing sequence of Input array of active nodes.

JavaScript code to form an adjacency list:

```
list_of_nodes[pairs[i][0]].adjList.push(list_of_nodes[pairs[i][1]])
list_of_nodes[pairs[i][1]].adjList.push(list_of_nodes[pairs[i][0]]);
```

Recursive DFS with Connected Component search algorithm: The aim of this algorithm is to compute all the cluster of connected subgraph is part of the static topology. These connected topologies will have information of nodes that connected and a subgraph number that will be an identifier for the connected fragmented network.

In implementation context, this algorithm was divided in to two parts searching and numbering the executed loop. The adjacency list is input to this execution process which will sequentially scan all the nodes present in the list. The output is a List data structure of all the identified nodes, marked with subgraph number consisting of all the nodes belonging to that subgraph. An adjacency List of the entire marked element is computed as an input data to the subgraph visualization.

Algorithm for Connected Components: [28]

1. Search for the unvisited node index.
2. Mark the node as visited.
3. Follow the next node in adjacency list of the previously marked node.
4. If the node is unmarked repeat step 2 else follow step3.

5. If all the nodes in a adjacency list are marked visited increase the graph no. counter and repeat step1 until are the nodes are marked visited.

Pseudo code:

```
InputData = [] // Array of active elements extracted from dump files

SourceArray = [array of source node]
TargetArray = [array of target nodes]
Create adjacencyList from Pairs
foreach (elements in InputData[i])
{AdjacencyList(sourceArray [i]) = AdjacencyList(targetArray [i])
};

CC_No = 1;

DFS (node);
check node [InputData[i]];
mark InputData[i] = visited;
check next node form adjacencyList
If
(nextnode = = unvisited ;)

Then DFS (nextnode)
else
CC_No++;
Create adjacencyList for each (InputData[i]== visited);Visualize subgraph;
```

d. Subgraph Visualization from CC Algorithm

A very similar approach from Static Graph, this visualization is only concern with the active elements which were identified from Connected Component algorithm. The subgraph is displayed over as island of connected trees. The data structure for the algorithmic subgraph view is also similar to the Static visualization. The union pair is extracted from the same JavaScript Object with the both the node shall present in the applied algorithm's list of nodes.

In addition to this, the topologies are concerned only with the connection to the Nodes. Therefore, the inclusion of Breakers does not count in this implementation. The code that's filters out node with string including characters "BRK" enlisted below.

```
nodes.filter(function(d) {
var name = d.name;
```

```
if (name.indexOf("BRK") > -1){ {return 0} }
Else {return 1 ;}
```

5.5 Results

The application was tested on a browser that supported HTML-5 webpages. Two separate webpages were deployed to obtain result of stated objectives i.e. the Visualization of static topology and Visualization of algorithmic connected topologies.

The individual webpages were provisioned with input box to enter a Timestamp that will be an argument for the reading the data from dump files. The output visuals were needed to be refreshed with a new entry in timestamp argument box.

5.4.1 Static Graph Visualization

Coming on to the static visualization the Force directed layout presented sanity in visualizing the graph. The colouring of the active nodes and link between active nodes clearly gives a good institutive idea on how the topologies are formed and the active loads in the network. For a specific time instance the application retrieved all the breaker status successfully, buses are considered "always active" nodes and loads status was derived from active status of respective breakers.

In Fig. 5.2 as shown, with a differentiated clamping to particular nodes, in this case for Buses and Loads, provide with logical understanding of an electrical grid. With Loads placed outwards and the Buses Inwards gives a less congestive view. It illustrates an intuitive visual in understanding the grid status for a specific time instance.

Fig 5.2 Complete SYSLAB Grid – Static topology Visualization.

5.4.2 Algorithmic version of Graph (Subgraph Visualization)

The connected topologies were displayed only with Buses and Loads that describes the consumption aspect of the grid. The adjacency list output of all the identified formed the data structure input to the subgraph visualization .The subgraph achieved was identified with a unique serial number for number of such subgraph. The final result obtained was fragmented topologies as loosely connected buses and loads of floating small graphs. As in Fig. 7.2, clearly illustrates the island of subgraph which can also be considered as fragment of complete Syslab topology obtained in Fig 5.3

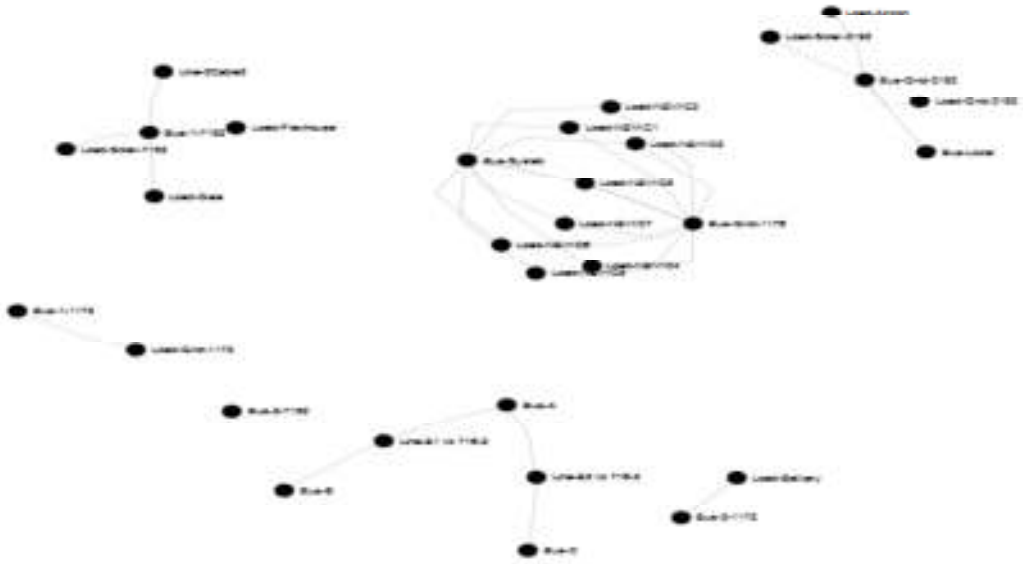

Fig 5.3 Algorithmic view of connected components

In Fig. 5.3 above, a cluster of connected topologies resulted from the Connected Component Algorithm, a simplified visualization from the static topology that is concerned with only representing Buses (power source) and consumption load devices.

5.6 Discussion and Future Work

The main objective in this application was to imply ideas of graph theory to electrical grid network, which were accomplished in a certain predetermined phases. The whole application development was divided into four phases of development that included reading the data file, creating a static topology, application of CC algorithm and visualizing the subgraph.

1. Database for Links and Nodes

Data source for the information of nodes and all the present linkages between all the nodes were manually were stored in Static file in JSON format. For every modification in a Syslab grid network, addition and deletion of the new buses, breakers or loads needs a manual entry to this static file.

Dynamic or online entry to this process could facilitate the user experience in contrast to going offline for every small change in the network.

2. D3 Library for Data and Visualization

A very straight application of force layout on visualizing the graph with nodes and edges is clearly seen in this application. Nodes from the JSON object were created as SVG element in HTML webpage with every tick function execution, i.e. with every tick event the nodes were evaluated with attributes predefined and the link defining the connections between nodes computed with the length depending on the placement of the nodes. Nodes were placed arbitrary on defined SVG canvas, so the link distance is not a static property for all the links. Negative charge of high value (-300) and large link length gave a good separation between the nodes which also provides clear vision in understanding the network.

Variable attribute for every link would have provided a breakthrough in achieving most of visualization objectives. Clamping of nodes approach was adopted instead, but that too had limitations avoiding crossing links. Aligning Buses nodes inwards gave a visual understanding of placement of the power source and loads on the extreme right or left of respective buses gives tree architecture.

As the name suggest all the data and visualization aspects in application development were accomplished using API supported by the library.

3. DFS search in CC algorithm

Depth first search method proved successful in enumerating all the nodes (including buses, breakers and loads) with the recursive search method. The algorithm had a fast process time for finite number of nodes in this scenario for Syslab grid. The subgraph achieved was identified with a unique serial number. Which in future a Map structure or hash Map data structure can hold all the unique topologies generated in past. It will be a catalyst in understanding of the particular grid network; blackouts and grid failure can be figured out with the aid of such study and information.

For a highly huge number of nodes, the recursive algorithm might not be a preferable solution. In this case use of Stack might be efficient with a numerous Pop and Push operations, which in contrast also increase the processing time for whole network.

4. Future improvements

The current development can be improved with following additions in the project:

1. The current application can further extended to display auxiliary electrical information of the particular grid. Voltage, Current, Kwh, Kvah, frequency etc. data can be incorporated in a detailed view for grid.
2. Optimization of grid would be good way to extend this application, where the grid status would be able to learn from the topologies occurred in past. A best possible solution to connect a distant load in the grid would be computed by using machine learning algorithm.
3. Control of the grid is an effective application on scale on automatizing the grid circuit. Being able to communicate efficiently with the control mechanism will certainly add to autonomous system development.
4. Application in Energy Management system (EMS) where the electrical connections are presented as Single Line Diagram and monitored for optimization of the present electrical architecture. This visualization shall reduce the stress on monitoring of unwanted electrical equipment and assist in Supervisory control system.

6. CONCLUSIONS

The main focus in the application design was to foresee the opportunities offered by the Grails application and not to develop a ready to deploy web application.

- This research study was based on developing a test web application that could be deployed on a Web container or a Servlet container.
- A multi-tenant application is proposed in integration of MES and ERP with a view of incorporating IoT technology.
- MES database with product list generated by provider enterprise able to be exposed over a cloud platform that can be accessed by the ERP applications of partner clients through Grails framework.
- Tenant entities that are provider MES and client ERP have an access to the service embedded with Grails application development for data persistence and data customization on schema level.
- Class loading mechanism of Java/Groovy is the pivot of tenant security through secured class loading mechanism complimented by Spring Security plugin.
- Grails artefacts (controllers, service, domain etc.) were easy to build through a standard script that manages required configurations on servlet built up; simply placing the appropriate artefacts over the relevant folder marked the identity for the artefacts enabling faster development of the web application.
- Scaffolding of the controllers relieved us from need of building simple user interface explicitly for the GORM domain controller i.e. Base Material in our case.
- In the Visualization context of IoT approach manufacturing data and business systems we have put forward a flexible substitute for conventional HMI system that are limited to vicinity of manufacturing facility.
- With the help of modern open source library it is possible to build an interface to supervise SYSLAB grid. Real time and historical information in form of graph of nodes and links possible to publish on web application using JavaScript bas D3 library and accessed through modern HTML5 supported browsers.
- D3 supports variety of visualization algorithm apart from “Force-Directed” one used in the method described here that can be useful in building various KPI (Key Performance Indicators); which can be important factor in Big Data analytics for Industry 4.0.

7. REFERENCES

[1] ""FAO - News Article: 2050: A Third More Mouths To Feed". *Fao.org*. N.p., 2016.

[2] John, Yongjean and Ki-Heung Yim. "A Study On An Environment Of ERP Introduction". *IEEE* 6 (2001): 84 - 89 vol.6.

[3] Van der Linden, Dirk. *How To Apply ANSI/ISA S88 (America) Or IEC 61512 (Europe)*. 1st ed. Hageschool Antwerpen. Print.

[4] Theron, John. *ISA-95: A Foundation Model For Business Intelligence For Manufacturing*. 1st ed. incuity, 2008.

[5] Scholten, Bianca. "Integrating ISA-88 and ISA-95". *Isa.org*. N.p., 2016.

[6] Fu, Ruixue et al. "Research On Integrated And Flexible MES Based On Agency And SOA". *2008 International Symposiums on Information Processing* (2016): n. pag. Print.

[7] Soplop, Jeffrey et al. "Manufacturing Execution Systems For Sustainability: Extending The Scope Of MES To Achieve Energy Efficiency And Sustainability Goals". *2009 4th IEEE Conference on Industrial Electronics and Applications* (2009): n. pag.

[8] MOHAJERI NARAGHI, ASHKAN. *Software Selection For Using Manufacturing Shop-Floor Data In MPCS*. Gothenburg, Sweden: CHALMERS UNIVERSITY OF TECHNOLOGY, 2011. Print.

[9] European Parliament's Committee on Industry, Research and Energy (ITRE),. *Industry 4.0*. Brussels: N.p., 2016. Print.

[10] "The DIKW Pyramid". *The Answer is 42. On Data, Information and Knowledge*. N.p., 2015.

[11] Namboodri, Chet. "Making Smarter Manufacturing and Iot A Reality Today". *blogs@Cisco - Cisco Blogs*. N.p., 2016.

[12] "Understanding the Cloud Computing Stack: Saas, Paas, Iaas". *Support.rackspace.com*. N.p., 2016.

[13] "What Is Iaas?". *Interoute*. N.p., 2016.

[14] "I Dream Of Iot/Chapter 4: Iot And Cloud Computing - Wikibooks, Open Books For An Open World". *En.wikibooks.org*. N.p., 2016.

[15] Tuley, John. "Introduction To Designing Multi-Tenant Web Applications - Quick Left". *Quick Left*. N.p., 2013.

[16] Application, Architecting and +Raji Sankar. "Architecting A Multi-Tenant Application". *Java Code Geeks*. N.p., 2016.

[17] "Introduction to Java Web Development - Tutorial". *Vogella.com*. N.p., 2016.

[18] Ledbrook, Peter and Glen Smith. *Grails In Action*. Shelter Island, NY: Manning Pub., 2014. Print.

[19] "Class Loading In Java/J2EE". *Learn how to learn....* N.p., 2010.

[20] Namboodri, Chet. "Making Smarter Manufacturing and Iot A Reality Today". *blogs@Cisco - Cisco Blogs*.

[21] "Spring MVC Framework Tutorial". *www.tutorialspoint.com*.

[22] *Top 5 Considerations When Evaluating Nosql Databases*. 1st ed. mongoDB, 2015.

[23] "GORM for Mongodb 5.0.6.RELEASE". *Gorm.grails.org*.

[24] "Grails Spring Security Core Plugin". *Grails-plugins.github.io*.

[25] "Spring Security". *Projects.spring.io*.

[26] "2 Scheduling Basics 1.0.2". *Grails-plugins.github.io*.

[27] Bob Bockholt and Paul E. Black, "adjacency-list representation", in Dictionary of Algorithms and Data Structures [online], Vreda Pieterse and Paul E. Black, eds. 14 August 2008

[28] Eppstein, David. "ICS 161: Design And Analysis Of Algorithms". 2015. Lecture.

APPENDIX

List of abbreviations

The following table describes the significance of various abbreviations and acronyms used throughout the thesis.

Abbreviation	Meaning
MES	Manufacturing Execution System
ERP	Enterprise Resource planning
IoT	Internet of Things
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
GORM	Grails Object Relationship Management
CRUD	Create Read Update and Delete
HTML	Hyper Text Markup Language
RDBMS	Relational Database Management System
SQL	Structured Query Language
CC	Connected Components
DFS	Depth First Search
JSON	JavaScript Object Notation
ISA	Internal Society of Automation

List of Figures

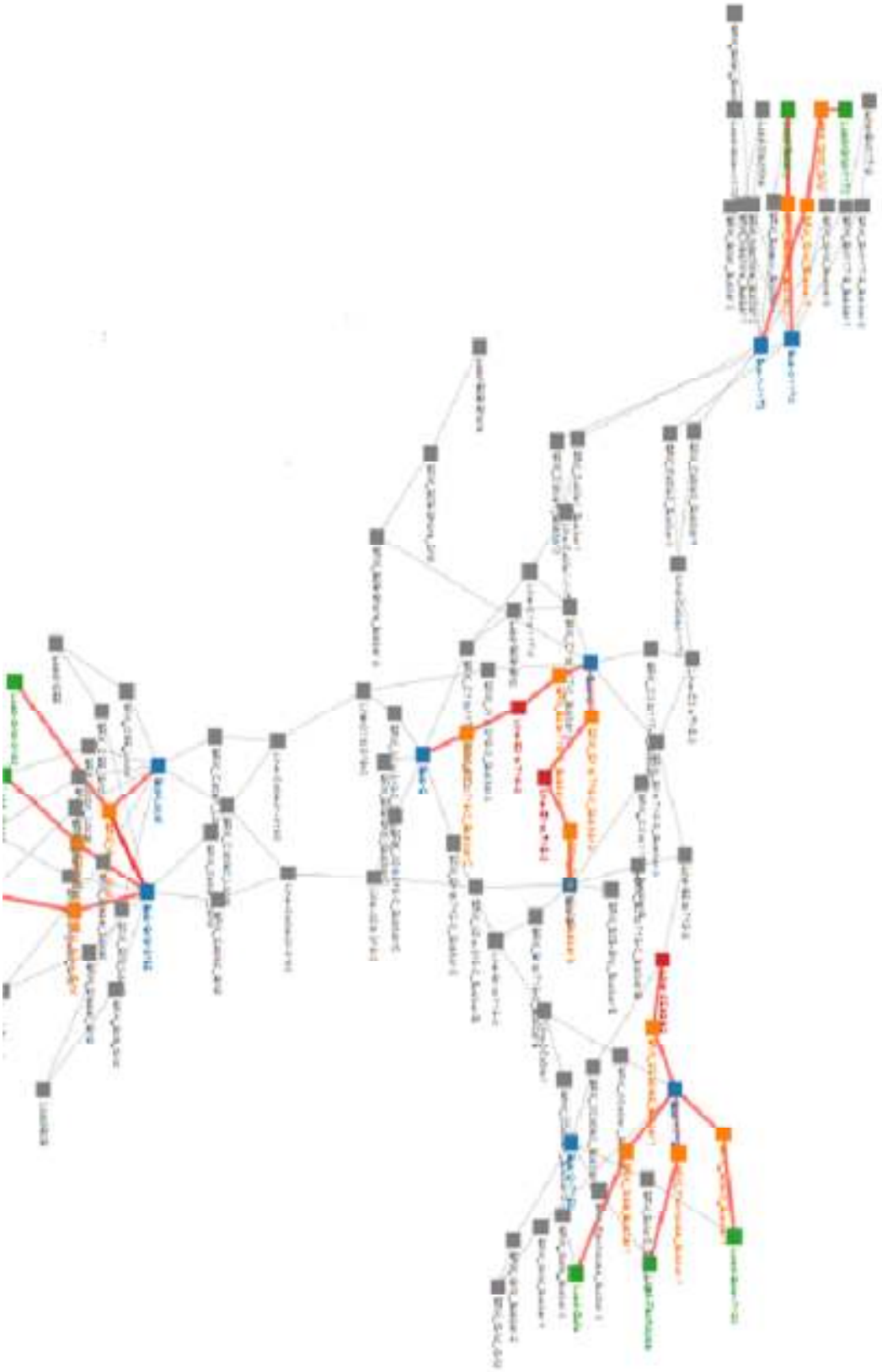

Fig.1 Static topology of SYSLAB network

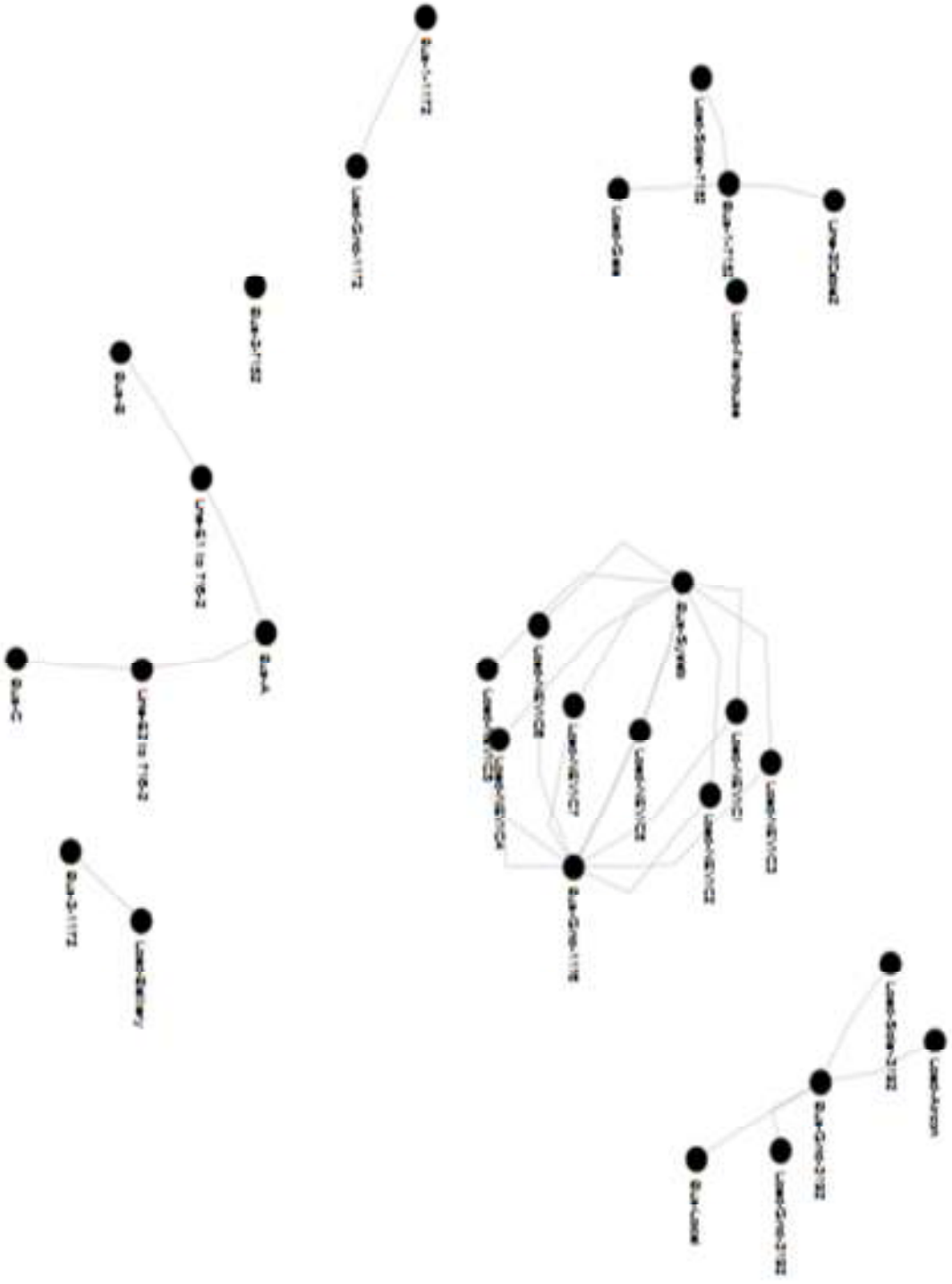

Fig. 2 Connected component algorithmic view of SYSLAB network

Development Code

Pseudo code implementation of CC algorithm using DFS search

```
InputData = [] // Array of active elements extracted from dump files

SourceArray = [array of source node]
TargetArray = [array of target nodes]
Create adjacencyList from Pairs
foreach (elements in InputData[i])
{AdjacencyList(sourceArray [i]) = AdjacencyList(targetArray [i])
};

CC_No = 1;

DFS (node);
check node [InputData[i]];
mark InputData[i] = visited;
check next node form adjacencyList
If
(nextnode = = unvisited ;)

      Then DFS (nextnode)

      else

      CC_No++;

      Create adjacencyList for each (InputData[i]== visited);Visualize

      subgraph;
```

Base Material Domain Class

```
package com.app
import org.bson.types.ObjectId
class Material {
        String matNr
        String matName
        String b1
        String b2
 static constraints = {
  }
        static mapping = {
                table 'material'
                id column: '_id', generator: 'assigned', name : 'matNr', type : 'String'
                //id composite : ['_id','matNr']
        }
        }
```

Base Material controller

```
package com.app
import grails.plugin.springsecurity.annotation.Secured
import static org.springframework.http.HttpStatus.*
import grails.transaction.Transactional

@Transactional(readOnly = true)

@Secured('ROLE_FRAMEWORK_USER')
class MaterialController {

 static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]

 def index(Integer max) {

    params.max = Math.min(max ?: 10, 100)
    respond Material.list(params), model:[materialInstanceCount: Material.count()]
  }

  def show(Material materialInstance) {

    respond materialInstance
  }


  def create() {
    respond new Material(params)
```

```
    }

    @Transactional
    def save(Material materialInstance) {
        if (materialInstance == null) {
            notFound()
            return
        }

        if (materialInstance.hasErrors()) {
            respond materialInstance.errors, view:'create'
            return
        }

       //materialInstance.save flush:true
                        materialInstance.insert() //(failOnError: true)
        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.created.message', args: [message(code: 'material.label',
default: 'Material'), materialInstance.matNr])
                redirect(url: "http://localhost:8080/multitenancy/material/index")
            }
            '*' { respond materialInstance, [status: CREATED] }
        }
    }

    def edit(Material materialInstance) {
        respond materialInstance
    }

    @Transactional
    def update(Material materialInstance) {
        if (materialInstance == null) {
            notFound()
            return
        }

        if (materialInstance.hasErrors()) {
            respond materialInstance.errors, view:'edit'
```

```
            return
        }

        materialInstance.save flush:true

        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.updated.message', args: [message(code: 'Material.label',
default: 'Material'), materialInstance.id])
                redirect materialInstance
            }
            '*'{ respond materialInstance, [status: OK] }
        }
    }

    @Transactional
    def delete(Material materialInstance) {

        if (materialInstance == null) {
            notFound()
            return
        }

        materialInstance.delete flush:true

        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.deleted.message', args: [message(code: 'Material.label',
default: 'Material'), materialInstance.id])
                redirect action:"index", method:"GET"
            }
            '*'{ render status: NO_CONTENT }
        }
    }

    protected void notFound() {
        request.withFormat {
            form multipartForm {
```

```
            flash.message = message(code: 'default.not.found.message', args: [message(code: 'material.label',
default: 'Material'), params.id])
            redirect action: "index", method: "GET"
        }
        '*'{ render status: NOT_FOUND }
    }
        }
def savefrmprovider(){
 def newNO = params.matnr
            [matnr: newNO]

 //                 redirect action : "create", params : [matNr : newNO, matName : "null", provider_code :
"null", manufacturer : "null" ]

                   def baseInstance = new Material(matNr : newNO, matName : "null", provider_code : "null",
manufacturer : "null" )

                   baseInstance.insert()

                   render 'saved'   }
 }
```

Tenant Specific class definition

```
package com.tenant2
import org.codehaus.groovy.grails.web.context.ServletContextHolder
import org.codehaus.groovy.grails.web.servlet.GrailsApplicationAttributes
import groovy.json.*
import groovy.lang.Delegate
class ExtendedMaterial   {
// @Delegate Base newbase = new Base()
String matNr
//String tenant1name
//int tenant1ID
//String matNames
//String nparam
//String newmat2
static int i = 0
```

```
String additionalInformation1;
 def printclient(tenant1ID){
        println "Client code printed from tenant1 $tenant1ID"

        }
        public static update() {
        def context =
ServletContextHolder.servletContext.getAttribute(GrailsApplicationAttributes.APPLICATIO
N_CONTEXT)
   def providerService = context.providerService
            ExtendedMaterial x = new ExtendedMaterial();
                x.matNr = String.valueOf(i)
                x.additionalInformation1 = "addn attr: " +String.valueOf(i)
                println 'additional matNr5: ' +x.matNr
                i++
           // Datasource for MongoDB
           def json = JsonOutput.toJson([Materialno : x.matNr , addnattr :
x.additionalInformation1 ])
                def matno = x.matNr
                providerService.mongoTfr(json, matno ) } }
```

Service Handling of Tenant specific class

```
package multitenancy
import base.Scheduled;

class TestJob {
        def providerService
   static triggers = {
   simple name: 'mySimpleTrigger', startDelay: 5000, repeatInterval: 10000
  }
  def group = "MyGroup"
```

```
def description = "Example job with Simple Trigger"




def execute(){
  print "Job run!"


        //println providerService.providergbvar

        if(providerService.providergbvar){
        Scheduled.updateables.every { entry ->
                entry.key."$entry.value"()
                println Scheduled.toupdate }
                }  }}
```

Service Class for Tenant

```
package com.app
import org.springframework.web.context.support.WebApplicationContextUtils
import java.util.concurrent.*
import javax.annotation.*
import grails.transaction.Transactional
import auxillary.ClassBuilder
import groovy.lang.Closure;
import groovy.lang.GroovyClassLoader;
import auxillary.StreamPrinter
import com.mongodb.Mongo
import com.mongodb.MongoClient
import com.mongodb.client.MongoCollection
import com.mongodb.util.JSON
import com.mongodb.DBCollection
```

```
@Transactional
class ProviderService {
 Class providerClass
 ClassLoader parent = Thread.currentThread().getContextClassLoader()
 def gcl = new GroovyClassLoader(parent)
 def gcl2 = new GroovyClassLoader()
 def gcl3 = new GroovyClassLoader()
 def url  = new File("C:/Users/ara/Documents/workspace-ggts-
3.6.4.RELEASE/multitenancy/mdefinition/tenantdef").toURI().toURL()
 ExecutorService executor = Executors.newSingleThreadExecutor()
 def TestJob
 def providergbvar
 def baseService
 static scope = "singleton"


   def runscript(def filepath){
         println filepath
         def builder = new ClassBuilder(gcl3)
   def binding = new Binding()

         binding.setVariable("builder", builder);

def engine = new GroovyScriptEngine("c:", this.class.classLoader)


  engine.run(filepath, binding)

  println gcl3.getLoadedClasses()  }


def fileM(){

         executor.execute{
         String fChanged
```

```
        def appPath =  System.getenv('GRAILS_APP')

         appPath = new File( appPath ).absolutePath
         def binding = new Binding(fdiff : fChanged)
    def engine = new GroovyScriptEngine(appPath)
    engine.run("scripts/dirfil.groovy", binding)

        }

        }
@PreDestroy
void shutdown() {
  executor.shutdownNow()
}



def loadclass(){
                //providerService.loadclass()
                def ClassLoader parent = Thread.currentThread().getContextClassLoader()
                gcl.clearCache()
                 providerClass = gcl.parseClass(new File
("C:/Users/ara/Documents/workspace-ggts-
3.6.4.RELEASE/multitenancy/mdefinition/tenantdef/provider_admin.groovy"))
                 base.Scheduled.addScheduable(providerClass, "update")


                //gcl.addURL(url)
        //      gcl.loadClass("C:/Users/ara/Documents/workspace-ggts-
3.6.4.RELEASE/multitenancy/mdefinition/tenantdef/provider.groovy")
                 println 'Provider class reloaded'
                 println gcl.getLoadedClasses()
                 println "class loaded in gcl: "+gcl.getLoadedClasses()
                 println "tenant class object: "+providerClass
```

```
                def f  = providerClass.newInstance()


                println f.metaClass.respondsTo(f,"update") ? "Exists: MethodExists()" : "Not
Exists: noMethodExists()";
                if (f.metaClass.respondsTo(f,"update")){ providergbvar = true }
                else {providergbvar = false}
        }

def recompile(){
def clsname =  gcl.recompile(url,providerClass.getName(), providerClass)
println clsname
gcl.getLoadedClasses()
        println 'class recompiled'



        }
def instantiate(){


        def pvrInstance = providerClass.newInstance()

        pvrInstance.matnr = '31'
        pvrInstance.matNames = 'provider 31'
        pvrInstance.newmat1 = 'new attr'
        pvrInstance.additionalInformation5 = 'addn atr'
        println pvrInstance.matNames
        println pvrInstance.additionalInformation5
         println pvrInstance.newmat1
        }

def mongoTfr(def json, def matId) {
```

```
                MongoClient mongoClient = new MongoClient()


    def db = mongoClient.getDatabase("foo")

        MongoCollection collection = db.getCollection("provider")

                collection.insert(JSON.parse(json))

                println 'tfrd to mongodb'
                baseService.save(matId)
}

def test(){
        println 'service called frm POGO'
        }
}
```