Publish and Subscribe Pattern for Designing Demand Driven Supply Networks

David R. Gnimpieba Zanfack, Ahmed Nait-Sidi-Moh, David Durand, Jérôme Fortin

Publish and subscribe pattern for designing Demand Driven Supply Networks

David R. Gnimpieba Z.[1,2]
Ahmed Nait-Sidi-Moh[1]
David Durand[2]
Jérôme Fortin[1]

University of Picardie Jule Verne (UPJV),
[1]Laboratory of Innovative technologies (LTI), 48 Rue Raspail, 02100 Saint Quentin, France
[2]Laboratory of Modeling, Information and Systems (MIS), 14 Quai de la Somme, 80080 Amiens, France

Abstract. Logistic flows, business process management and collaboration remain a major problem in the supply chain. In this article we are going through this issue to propose an integrative and collaborative approach. More precisely, we develop a cloud-based and service-oriented bus for business interoperability for logistic flows. Though the bus, we define protocols and standards allowing the integration of data formats which are for the most proprietary and heterogeneous. The bus also allows data sharing between processes and actors involved in the flow. Key features of this bus are event handling and processing from the physical flow and real-time notification to stakeholders.

Keywords: Internet of things, Collaborative platforms, Demand Driven Supply Network (DDSN), Enterprise Service Bus, Publish/Subscribe pattern.

1. Introduction

Business process collaboration is one of the main problem enterprises are facing today. The reason is that each enterprise has its own business standards and Information Systems (IS) and its own infrastructures. Also, data format and communication protocols between supply chain actors are largely heterogeneous. With the development of Internet of Things (IoT), another problem is how to securely collect and make available real-time events, data and information from supply chain objects or physical flows. The scientific problem stated here is the integration of multiple technologies in a cloud-based bus to enhance collaborative supply chain inspired by DDSN strategy.

We suggest an integration approach of the Supply Chain IT Communication Infrastructure, based on cloud service bus and Service Oriented Architecture principles to facilitate interoperability and data sharing between business actors. We also are dealing with Internet of Things concepts, FI-WARE middleware. We use a complex event processing engine to handle events and flows of data from supply chain objects and services in real time, process and notify them. The engine allows data and information availability and sharing, event triggering and processing and leads to collaborative supply chain, improves responsiveness and fast decision making. It also facilitates business interoperability, a significant challenge for the innovative supply chain and business process management.

2. Overview

In this section, we give an overview of basic concepts and technologies of business process management and collaboration.

Scientists have already proposed solutions based on the integration of technologies to meet the requirements of collaborative supply chain [17, 18]. Most of these solutions are inspired on Enterprise Service Bus (ESB), web portals coupled with database server (DBMS), or ERP's model. In these cases, SC partners are connected to the gates and have a view of the logistics unit for which they have access right [16]. For models based on web services, each actor in the supply chain requests web operations from its collaborator to get the desired information. This situation creates a mesh of point-to-point communication architecture between SC partners. The problem here is how supply chain partners will share data with all their partners, how to deal with confidentiality, security, access right, service level agreement, and reliability of the collected data, how to merge proprietary protocols to a common uniform protocols among partners. Many existing platforms didn't have a cloud-oriented data storage strategy (*i.e.* NoSQL, cloud storage, AWS). Moreover, these architectures rarely mention the notion of notifications and real-time event processing. How to enable real time data access from everywhere (PC, tablet, iPhone), from any platform (Windows, Linux, IOS, Android) and anyhow while ensuring the same level of security, reliability and availability? All these questions lead to rethink about existing solutions based on ESB integration, enterprise websites, ERP's and WMS. To sum up, we could say that one solution to address DDSN strategy is to propose a cloud base service bus as a middleware for the overall supply chain collaboration purpose. The given bus has to enable data and event sharing and notification between supply chain actors and could be based on Publish/subscribe pattern, Event Driven Architecture and all the above mentioned technologies as a response to the limits of existing solutions and platforms.

- **Internet of Things:** the IoT is an evolution in computer technology and communication that aims to connect objects together through the Internet. By object we mean everything that surrounds us and can communicate or not [1]. The main objective of the IoT is to make these objects more intelligent and communicating. The flow of information and events generated by the interconnection of these objects is used to facilitate their tracking, management, control and coordination, logistics flows in the supply chain [2]. The integration of heterogeneous technologies and concerns are some of main challenges to achieve in order to take advantages of this new paradigm [3].

- **Complex Event Processing (CEP):** Events are messages indicating that something has happened and could change the state of affairs [6]. Complex events are the combination of multiple events from heterogeneous sources, in a given time interval [5]. In Event Driven Architecture, Complex Event Processing consists of event handling, patterns matching, in order to produce result events or actions [7]. CEP could be used for collaborative business process, for example, initiating the order process to a supplier when a stock reaches some levels [4]. It can be used also for aggregating events and applying predefined business rules or patterns to extract key information for business analysis and for real time decision making. Another usage of CEP is business monitoring by transforming events into key performance indicators (KPI) [4].

- **Publish/ Subscribe patterns:** Publish/Subscribe pattern differs from other message exchange patterns because only one subscription allows a subscriber to receive one or more event notifications without sending request to service producer [8]. Publish/Subscribe pattern seems to be the right candidate for processing events in the context of multiple event producers and consumers using several heterogeneous sources [8]. The pattern can be used in business process management systems where a customer could act as a service by subscribing to supplier service and publishes orders or inputs. The supplier service sends notifications event to customer service [9].

- **Service Bus:** service bus is an evolution of Service Oriented Architecture (SOA) in the field of Enterprise Architecture Integration (EAI). Among other enterprise system architectures, the Enterprise Service Bus (ESB) provides loose coupling, reliability and large flexibility. Furthermore ESB facilitates the interactions of services and applications, business activity cooperation and interoperability regardless on their heterogeneous protocols, data sources and format [10], [11]. Publish/Subscribe pattern from Event Driven Architecture enhances SOA: a service consumer can subscribe to one or more services once a time, making an “Advanced SOA”. The ESB main role is to provide interoperability by enabling consumers to call services providers supply [12]. Its features include connectivity, data transformation, intelligent routing, security, reliability, service management and administration tools.

- **Demand Driven Supply Networks:** Traditional supply chain and business management systems have lackness because suppliers didn’t have a global visibility on customers’ orders and market demand. DDSN is an IT approach for business-to-business collaboration and interoperability. DDSN recommends data sharing on inter-company supply chain. By applying this approach, instead of responding individually to isolated customers’ orders, it would be better if suppliers could reorganize themselves and work together by sharing more data in order to better respond all market demands [13]. DDSN uses the pull technique, i.e. the supply chain is driven by customers demand by reacting, anticipating, collaborating and orchestrating [14]. According to the scenario, the out of stock level (OOS) is about 8% and could go up to 30 %. Reliable information sharing could lower OOS-rates and improve Demand Chain Management [15].

3. Service bus Architecture

The figure 1 shows the global architecture of the developed bus. Event producers and consumers publish real time events through the cloud using the bus. Event subscribers are notified automatically, regardless of the protocols used. The features and internal structure of each component of the bus is detailed in the next paragraphs.

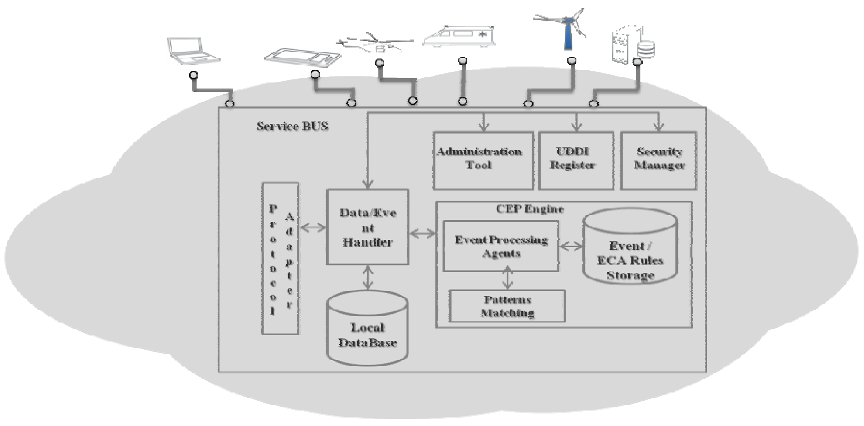


Fig. 1: Bus global architecture

3.1. UDDI Register

The UDDI Register is a directory for service registering, discovery and subscription. This component specifies how web services (WS) and other legacy applications register and what they provide. It can be business entities or business services (WS for instance). The protocol we describe here is based on the UDDI registry description, Google APIs Discovery Service and Google OAuth 2.0. We describe three basic methods as key functionalities of the service registry on the provider side, four operations on the client or service consumer side. These seven methods are key functionalities of our cloud Bus. Hereafter we detail each operation.

- *registerService(serviceUri, serviceName, serviceDescription, providerLogin, providerPwd):* This method allows a service provider to register to a service in the share registry. Here, we assume that the provider is already registered with an account *(providerLogin, providerPwd).* When the provider sends request for service registering, the Data/Event Handler forwards the request to the UDDI registry. The registry stores the information about the service provider: the Uniform Resource Identifier (URI) of the service, the name, the description, and the provider name. Then, the UDDI registry notifies the provider by sending the service Id and the response code (200). Otherwise, the error code 400 (*bad request*) is sent and the reason of the failure (network problem, wrong parameters…). After the service registration, it is made available for discovery and subscription. The Data/Event Handler is one main component of the bus and will be detailed in what follows.
- *unregisterService(serviceId,providerLogin, providerPwd):* To unregister a service, the provider must send a request with login and password and the service unique identifier (*serviceId*). After receiving the request, the Data/Event Handler notifies all the subscribers, shuts down the service and removes it from the registry.
- *updateContext(serviceId, providerLogin, providerPwd, contextParam):* When the service is registered, it can be updated when the context changes, using the *updateContext* method. When receiving this request, the data/Event Handler checks for service context and updates parameters accordingly. The request variable

contextParam is a map of (keys/values) pairs, where *key* denotes the name of the attribute, and *value* the new value of this attribute.

- *discoverService(filteringCriteria)*: This method is used to search services matching given patterns. The Event Handler uses the filtering functionalities of the service bus to find services that correspond to the criteria *(filteringCriteria)*.
- *subscribeService(serviceUri, subscriberLogin, subscriberPwd):* Service consumer can subscribe to a service by sending the *subscribeService* request to the Event Handler. Note that we didn't manage Service Level Agreement between the service subscriber and consumer. This functionality is not in the scope of this work. So we suppose that the subscriber has access rights to subscribe and consume the provided service. When receiving the request, the Event Handler checks for arguments to avoid bad URI, login or password. Then, the Event Handler registers the client as a service consumer and notifies the requester that everything is correct by sending the status code 200. Otherwise, the error code 400 is sent back to the client.
- *updateSubscription(srviceUri, updateParams):* When subscriber information changes (login or password), the *updateSubscription* request is sent to the Event handler which updates the local database accordingly.
- *unsubscribeService(serviceUri, subscriberLogin, subscriberPwd)*: Consumer can request for service *unsubscription*. The Event Handler will remove it from the list and will no longer receive notification when the service context is updated.

3.2. Data/Event Handler

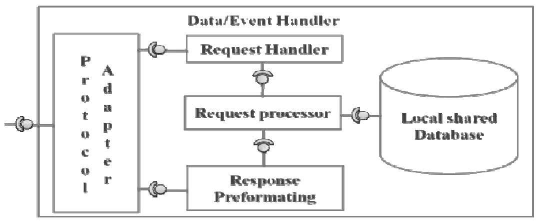


Fig. 2: *Data/Event Handler architecture*

The Data/Event handler is one of the main parts of the bus. It aims at managing data/event handling, the user's database for bus administration and security purposes. The data model behind this component is based on the FI-Ware business entity model.

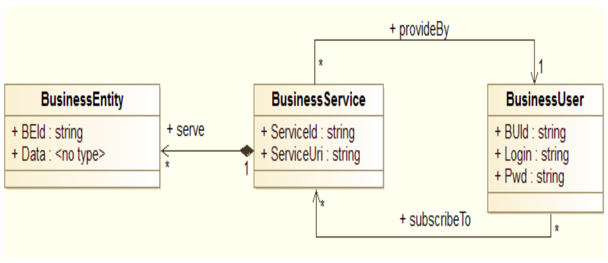
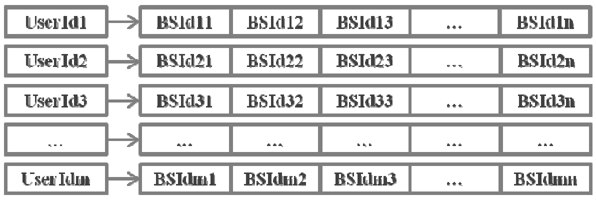


Fig. 3: *Internal Data structure*

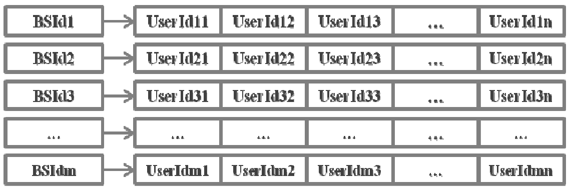
The business entity denotes a service, IoT objects serving their context or legacy system (ERP, WMS…) sharing objects from their internal database. Our data model has three main concepts and three relationships. The first concept is the BusinessEntity (BE). The Business Entity represents a business object (pallets, goods, transport facilities, container…), as well as any object in the flow of goods. A Business Entity has a unique identification number in the platform (BEId) and a collection of key/values pairs representing data related to the BE. Business entities can be served by business service (BS). Let's consider a web service providing goods or transport facilities or just data from a proprietary database. This service (BS) acts as an interface for this business entity to be well managed and shared by the bus. The Business Service has a unique identifier and a URI for binding. The Business User (BU) is an organization sharing data and service for collaboration purpose. BU can be also an operator hired in the flow of goods that needs information for business coordination. The user has a unique identifier, a login and password for security and bus administration. The service bus also manages a hash-map for storing services provided by business operators. For each business operator, the bus stores a *userId* as an entry key and a list of business service Identifier (BSId) as values (see Fig. 4).



UserId1	→	BSId11	BSId12	BSId13	…	BSId1n
UserId2	→	BSId21	BSId22	BSId23	…	BSId2n
UserId3	→	BSId31	BSId32	BSId33	…	BSId3n
…	→	…	…	…	…	…
UserIdm	→	BSIdm1	BSIdm2	BSIdm3	…	BSIdmn

Fig. 4: *Hash map for providedBy relationship*

The subscription relationship allows to know who subscribes to a given business service, and notifies them when necessary. This relationship is also implemented by a hashmap structure to facilitate access and reduce search time. In this map, keys are business service identifier and values are a list of service subscribers represented by their UserId, as illustrated in Fig. 5.



BSId1	→	UserId11	UserId12	UserId13	…	UserId1n
BSId2	→	UserId21	UserId22	UserId23	…	UserId2n
BSId3	→	UserId31	UserId32	UserId33	…	UserId3n
…	→	…	…	…	…	…
BSIdm	→	UserIdm1	UserIdm2	UserIdm3	…	UserIdmn

Fig. 5: *Hash map for Service Subscription relationship*

The last relationship allows a service subscriber to access business entities. Using that, the list of business entities served by business operators is stored temporarily.

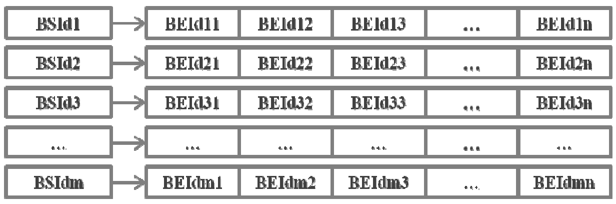


Fig. 6*: hash map for ServeBy relationship*

3.3. Protocol adapter

The protocol adapter is the intermediate layer between the bus and provider/consumer of services. It is composed of multiple protocols transformation. It is designed to support a large part of Internet of Things and web services protocols. A client sends a request with its own protocol and the adapter transforms this request into a unique format. The bus protocol is independent from the request and should be understandable by all the bus users. Our choice is focused on the MQTT (Message Queue Telemetry Transport) protocol for several reasons: MQTT protocol supports publish/subscribe operations, TCP-based, asynchronous and payload agnostic. Furthermore, this protocol is build for the Internet of Things. Conversely, when the bus responds to a client, the adapter transforms the bus response into the client specific protocol. The Protocol Adapter architecture is given in the Fig. 7.

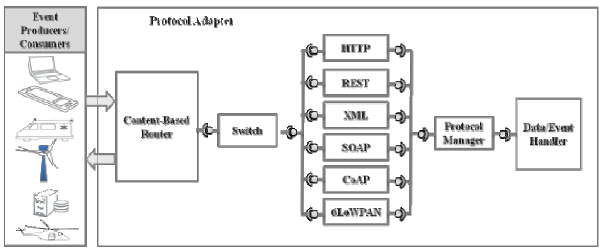


Fig. 7*: Protocol Adapter Architecture*

3.4. Complex Event Processing Engine

In this part, we use a CEP Engine to provide collaboration between customers and suppliers. Customer's orders are considered as complex events to be triggered by the CEP engine. The CEP engine analyzes each message, and notifies suppliers about the published order. When the supplier didn't have the product in stock, an event is generated by its front-end to the broker and then triggered by the CEP. When the CEP completing all lines Items, a notification event is sent to the customer.

As illustrated in the Fig. 8, the CEP module has three main components: the Event processing agent, the patterns matching and the local event storage database. When the events producers send requests to the service bus, after converting the request with the protocol adapter and making some security checks by the Event handler, it forwards the request to the Event processing agent. This component stores the event in the local database, and activates a pattern matching process. Pattern matching matches Event Condition Action rules (ECA) to event stream. When the event stream matches the ECA rules, the Event Processing Agent notifies business actors who registered the business rule in the database. To do so, the event storage has a Hash map that stores a set of associated ECA rules for each Business entity. For these last operations, we have identified three potential use cases.

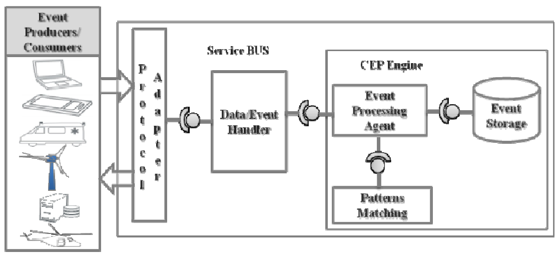


Fig. 8: *CEP Engine Architecture*

- ***1st use case*****:** *Count the customers orders in a certain time window* (e.g.: a week). If the total amount reaches a defined level or limit the bus notifies the provider of these goods. So the provider can start the manufacturing of goods or a delivery process if goods are available in stock. This leads to the following ECA rule:

```
When    customersOrderEvent
            Total = count (ROItems)
If        ((timeWindows) and (total> Limit))
Then        notifyProvider (total)
```

- **2nd *use* case:** We consider a case where the supplier has loyal customers. Customers and vendor registered earlier on the bus. Customers should notify supplier hotlines their level of stocks through the inventory management WS. When the aggregate amount of customer inventory reaches a defined level, the CEP must notify the supplier. After receiving this event the supplier can start a products manufacturing process, or undertake a procurement process for all affected customers.

4. Application to collaborative supply chain

- Event triggering for filling and shipping a container: The scenario we present here is the one where a container has to be filled and shipped. A Fourth Party Logistic (4PL) operator has a container to fill with a list of goods. The container is galvanized and shipped when all goods are filled in. When the container arrives at the distribution hub, goods are unbundled and delivered to end-customers via terrestrial transport operator. Goods come from different suppliers and operators whose transport modes are as different as end customers and are not in the same geographic area. The scenario shows the complexity of coordinating logistic flows and supply chain by triggering events generated by the flow of goods. We want to apply our cloud-oriented service bus architecture to manage and share information between all involved actors. The service bus (SB) also handles all events emitted by the flow of goods and the actors in real time. Furthermore, the bus notifies automatically all SC partners. This leads to facilitate the coordination and monitoring of the flow of goods. The CEP Engine incorporated in the SB help the 4PL operator to search service providers using the registry. Providers ensure the various phases of the process: filling the container, shipping, distribution of goods to the final client. We divide the process into three sub-parts: filling the container, shipping the container and goods delivery to end customers.

- **Delivery Sub-process:** When the container is unloaded from the vessel, unloading slip is signed and *containerUnloaded(containerId, boated,unloadingDateTime)* event is sent to the service bus. The container is transported to the distribution hub and *containerArrivedAtHub(containerId,DateTime)* event is sent. At the end, goods are distributed to end customers. When final customers receive their goods, the event *goodsReceived(goodsId, deliveryDateTime)* is sent to the 4PL operator. Finally, when all goods are received, the 4PL operator is notified by the event *allGoodsDelivered(DateTime)*, that ends the business collaboration process. The scenario we describe (Fig. 9) is represented with a BPMN diagram.

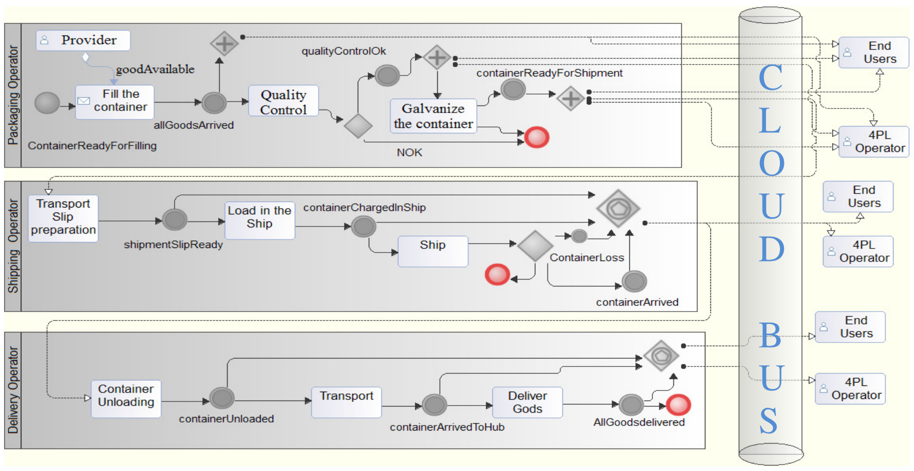


Fig. 9: *Shipping a container business process (with BPMN annotation)*

5. Conclusion

In this paper, we bring our contribution about the interoperability between business actors. We focus particularly on Demand Driven Supply Network as IT approach for business collaboration. Going through this issue, we propose a cloud-based, service oriented and event-driven bus, as we learn from Enterprise Integration Architecture. It hides protocols heterogeneity and multiple messages exchange format. As a result, it enables data sharing, service providing and subscription, real time event handling, filtering and automatic notification. Therefore, the solution enhances collaboration between stakeholders in the supply chain and improves logistic flows coordination.

Obviously, this architecture may be combined to cloud platform and big data, improving Business Intelligence and Business Activities Monitoring.

References

1. Sehgal, V. K., A. Patrick, L. Rajpoot: A Comparative Study of Cyber Physical Cloud, Cloud of Sensors and Internet of Things: Their Ideology, Similarities and Differences. IEEE Intern. Advance Computing Conference, pp.708-716, (2014).
2. D. R. Gnimpieba Z., A. Nait-Sidi-Moh, D. Durand, J. Fortin: Internet des objets et interopérabilité des flux logistiques: état de l'art et perspectives, UbiMob2014 : 10$^{\text{èmes}}$ journées francophones Mobilité et Ubiquité, Sophia Antipolis (France), http://ubimob2014.sciencesconf.org/40476/document, (5-6 Juin 2014).
3. Benghozi P.J., Bureau S., Massit-Follea F. : Internet des objets: Quels enjeux pour les Européens? Ministère de la recherche, Délégation usages de l'Internet, Paris, (2008).
4. M. Eckert, F. Bry: translation of "Aktuelles Schlagwort: Complex Event Processing (CEP)", German language in Informatik-Spektrum, Springer (2009).
5. Asaf Adi: Complex Event Processing, Event-Based Middleware & Solutions group IBM Haifa Labs (2006).
6. Vitra technology: Complex Event Processing for Operational Intelligence, A Vitra Technical White Paper, (2010).
7. M. Saboor, R. Rengasamy: Designing and developing Complex Event Processing Applications, Sapient Global Markets, (August 2013).
8. Arnon Rotem-Gal-Oz: SOA Patterns, Manning Publications Co., (Sept. 2012).
9. Wei Li, Songlin Hu, Jiantao Li, Hans-Arno Jacobsen: Community Clustering for Distributed Publish/Subscribe Systems, Cluster Computing (CLUSTER), 2012 IEEE International Conference, Beijing , pp: 24-28, (Sept. 2012).
10. Luis Garcès-Erice: Building an Enterprise Service Bus for Real Time SOA: A Messaging Middleware Stack, Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, Seattle, Washington, USA, 20-24 (July 2009).
11. G. Hohpe, B. Woolf: Enterprise Integration Patterns- Building and deploying Messaging solutions, Pearson Education, Inc., (10 Oct. 2003)
12. Nicolai M. Josuttis: SOA in practice The Art of Distributed System Design, O'REILLY Media, Pages: 344, (August 2007).
13. Enrique De Argaez: Demand Driven Supply Networks DDSN, Supply Chain report. [online] http://www.internetworldstats.com/articles/art087.htm. Last visit May 2015.
14. Martin R, GMA and AMR Research: The Demand Driven Supply Network DDSN, [online] http://www.internetworldstats.com/articles/art087.htm, (2014).
15. Gruen, T. W., D. Corsten and S. Bharadwaj: Retail Out of Stocks: A Worldwide Examination of Causes, Rates, and Consumer Responses, Washington, D.C.: Grocery Manufacturers of America, (2002).
16. Robert A. Davis. Demand-Driven Inventory Optimization and Replenishment: Creating a More Efficient Supply Chain. Book edited by Wiley, 2013. [Online] http://www.sas.com/storefront/aux/en/spscddior/66127_excerpt.pdf. (Last visit 05/2015).
17. Ariff, M.H., Ismarani, I., Shamsuddin, N., RFID based systematic livestock health management system. 2014 IEEE Conference on Systems, Process and Control (ICSPC), pp. 111–116. doi:10.1109/SPC.2014.7086240. (2014).
18. Tian-Min, C., Constructing Collaborative E-business Platform to Manage Supply Chain. International Conference on Information Management, Innovation Management and Industrial Engineering, pp. 406–409. doi:10.1109/ICIII.2009.255. (2009).