Volume 56, Number 1, 2015

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

ENTERPRISE BUS AS A SERVICE – AN IDEAL CLOUD CONNECTIVITY MODEL

Sorin POPA Mircea-Florin VAIDA
Technical University of Cluj-Napoca, 26-28 Baritiu str. Cluj-Napoca, Romania
coresystems ag, 69 Dorfstrasse Windisch, Switzerland
sorin7popa@gmail.com Mircea.Vaida@com.utcluj.ro

Abstract: Cloud computing is unquestionably one of the most important things happening in IT today. It's a concept meant to manage resources in a more responsible manner and to offer companies a professional, efficient way of accessing them. Nevertheless, one of the primary struggles of the cloud-oriented world is application integration. The primary goal of this project is to define an ideal connectivity model for moving the application integration from the organization side into the cloud. In this way, the organization will efficiently save resources: less deployment time, fast and clean new releases, no redundant infrastructure maintaining costs and of course higher productivity. As a lightweight and flexible solution, the "Enterprise Bus as a Service" architectural pattern, or short EBaaS, will manage everything needed by the organization to properly integrate their system into a one-click cloud application.

Keywords: *integration, connectivity model, cloud, ERP, EBaaS, ESB.*

I. EXECUTIVE SUMARRY

Cloud computing impacts the way classic organizations run in three major ways: First, organizations are no longer bound to use local software with expensive licenses, but can conveniently access cloud applications (SaaS), in an intuitive manner, via the Internet. Second, organizations are encouraged to use cloud platforms to develop and store their own applications and data. Finally, organizations can use private clouds, which are in essence cloud platforms used for internal, company-specific sensitive data.

The main challenge in adopting the cloud world for most of today's organizations, especially when dealing with a spread computational and data storage model, remains the integration. This refers to the way in which several applications, belonging to different platforms, maybe even different operating systems manage to work together and transmit data between each other[1][2]. The first approach in this case is usually point to point integration, but that quickly becomes messy and expensive to maintain. An alternative to this would be a SAO (Service-Oriented Architecture) suite, but this implies the allocation of new resources (in terms of time, money and human resources) often with poor results, leading back to point to point code. Another alternative lies in the possibility of going for an ESB (Enterprise Service Bus) integration platform, but this again would mean heavy deployments, unfamiliar development tools, management of legacy systems that are no longer supported by their original vendor and unjustified efforts from each and every organization trying to integrate their systems[3][4].

The aim of the present writing is to raise awareness of the importance and necessity of a generic integration connectivity model, highlight its requirements and considerations and emphasize on a number of must and nice-to-have features and functionalities of such a solution. This generic integration connectivity model will be furtherly referred to as "Enterprise Bus as a Service" or simply EBaaS.

Among the numerous benefits of such a solution, the main one, of course inherited from the SaaS concept itself, would be user-based pricing. Firstly, this would significantly decrease the financial risk of a company adopting this model, by minimizing the need of local owned resources, including servers and IT staff, and implicitly the upfront large investment requirement. On top of this, a major benefit is described by the freedom of the development team to prove the value of software, no longer bound to over or under-estimating the popularity of a software solution and developing plugged into a resource pool described mainly by abundance and elasticity, as using a server for a thousand hours or a thousand servers for one hour may make the difference. Finally, this would drive towards the extinction of the version-specific locked-in everlasting legacy systems, no longer supported by original vendors[5].

On the other hand, the requirements of such a still young concept are vast and various. One of the most important requirements maybe of cloud applications is trust. Cloud providers need to inspire it and prove that their services have good reliability, availability and security. Compared to a plain web application security model, the cloud adds an extra layer, that of virtualization, which guarantees the security of the application inside the cloud. Ensuring the privacy and security of such a solution requires the implementation of a fine-spun user policy which enables complete transparency to the user into monitoring their data sources, encryption and storage processes in the cloud. The crucial emphasize of this concept combines the virtualization security of the communication channel with

Manuscript received February 16, 2015; revised February 27, 2015

the failure recovery ability and a reliable user support model.

II. CONTEXT AND POSITION

Starting the early half of the past century, organizations kept pushing towards the development of software suites so called bill-of-material (BOM) processors and programs to manage inventory and develop material requirements plans. Purposes and capabilities of these software suites grew into numerous business function batches, such as CRM (Customer Relationship Management), SRM (Supplier Relationship Management) or ERP (Enterprise Resource Planner) of great importance to all modern organization today[6][7].

Customer Relationship Management (CRM), is a company-wide business strategy designed to reduce costs and increase profitability by solidifying customer satisfaction, loyalty, and advocacy. It is a widely implemented model that involves using technology to organize, automate, and synchronize business processes—principally sales activities, but also those for marketing, customer service, and technical support. Salesforce, Microsoft an Oracle are only a few of today's most popular CRM systems providers[6][8].

Supplier Relationship Management (SRM) is a comprehensive approach and a discipline of strategically planning for, and managing all interactions with third party organizations that supply goods and/or services to an organization in order to maximize the value of those interactions. It delivers a competitive advantage by harnessing talent and ideas from key supply partners and translates this into product and service offerings for end customers[8].

Short for Enterprise Resource Planner, ERP is a software systems that integrates internal and external management information across an entire organization and automates and facilitates the flow of data between critical back-office functions, embracing financing, distribution, accounting, inventory management, sales, marketing, planning, human resources, manufacturing, and other operating units[7][9]. Deriving from the less popular MRP (Material Requirements Planner), ERP systems initially focused on developing back office functions that did not affect customers. Later on, front office functions such as CRM, e–business systems or Supplier Relationship Management (SRM) were integrated, when the Internet simplified communicating with external parties.

Also known as "Enterprise Application Suite", the ERP concept has rapidly expended into almost all the business domains available. Among the most popular ERP systems worldwide we can name: Microsoft Dynamics, Oracle ERP and of course the very popular SAP[10].

Having an overview of all this concepts, strategies and approaches the application integration necessity forwardly emerged towards empowering interplay and communication between mutually interacting software applications in service-oriented architecture (SOA) throughout an organization. An Enterprise Service Bus (ESB) is fundamentally an architecture described by a set of rules and principles for integrating numerous applications together over a bus-like infrastructure. As a software architecture model for distributed computing it is a specialty variant of the more general client server software architecture model and promotes agility and flexibility, but first and foremost the ESB helps achieve efficiencies in skills, cost and time-to-value processes across your middleware solutions.

Given the current world concern for security and privacy, an ESB architecture must come with an integrated solution for ensuring security, from design, across all applications. These requirements for security and privacy can be summarized in the following properties: confidentiality, integrity, authenticity, non-repudiation and access control.

III. OBJECTIVES

The first objective of this project is to define and introduce an ideal connectivity model for the "Enterprise Bus as a Service" (or EBaaS, keeping the cloud-oriented terminology) as a standardized architectural pattern enabling previously discussed ESB software application suites to be delivered via SaaS.

This not-yet-matured concept already is a breakthrough in the enterprise software business, raising the barriers for user-based pricing licenses for every type of tool and application interoperability. More than that, this concept enables the externalization (out-sourcing) of the entire integration process; no longer will each and every organization have to build, deploy and maintain expensive integration software code bases. After a one time customization process, all this capability will stand one click away.

Implicitly, this concept significantly reduces the cost of this service. Delivering it via a multi-tenant SaaS, an EBaaS cloud application could theoretically serve an infinite number of customers, each benefitting of their own negotiated security and privacy.

Last, but definitely not least, we have to take into consideration upgrades. It is well known that companies fear software migration from one product to another, even one version to another, especially if we are talking about colossal systems like the ERPs. They can cause backward incompatibility, potentially introducing of new bugs in the system, deployment issues, ultimately time and money penalties. This concept will allow fast & clean deploy-free updates without even having to notify the client company about them.

As an important aspect worth mentioning in this context is that even though this research paper is not proposing a completely pioneer theory, this concept is still very young and yet far from maturing. The market already provides a bundle of connectivity-flavored cloud-based software integration services, but their concepts, architectural models and implementation design are almost always technology or product bound and have yet to improve towards a completely elastic and scalable integration model.

Mostly due to the undeniable global cloud migration trend, the interest shown already by multiple software provider companies in similar designs but also because of the significant importance of this cloud-oriented software architecture, a refined but mellow version of this concept will prove itself to be of great value and immediate interest to the enterprise software development industry.

IV. STATE OF THE ART

The idea of having a cloud-based ESB is now a popular debate for quite some time in the information technology world. Questions like "Can a cloud-based ESB really trim your integration costs?" are being asked more and more frequently and the general concerns are those that, by porting such a massive and complex software architecture to the cloud side, issues like operational support, security, latency and others may appear along the way[11][12].

Nonetheless, there are already a bunch of vendors and communities that released versions of such cloud-based-integration-platform flavored products. They come with various statements and application domains, under names like "Enterprise Service Cloud", "Cloud ESB" or "Cloud Hub", but have yet to mature and gain the market trust before they will be used on a significant market scale[13]. Among these cloud integration service busses we find:

- *StratosLive* the Java PaaS operated by WSO2. It is powered by the multi-tenant WSO2 Stratos cloud middleware platform and shares a common code base with WSO2 Carbon enterprise middleware platform. App Factory delivers a DevOps PaaS for building the digital business ecosystems.
- *Fiorano Cloud Platform*, which "(…) blends cloud-based integration, high-availability messaging and general business-process integration. The result is a cloud-based integration services environment that lets companies integrate cloud-based applications with on premise apps and resources", according to Atul Saini, Fiorano's CEO, [14].
- *Meritec SAAS Agile ESB*, a rapid application deployment environment/ tool services in local government to reduce costs and facilitate better service fulfilment.
- *MuleSoft CloudHub*, an integration platform as a service (iPaaS) that allows the deployment of cross-cloud integration applications, creation of new APIs on top of existing data sources, integration of on premise applications with cloud services, etc.

Some of the concerns and suppositions regarding this developing concept are true and of course some are not and this mostly depends on the way in which the EBaaS platform is used. The on demand version of the ESB architecture can be a bit more complicated but also more powerful when satisfying its on-premises needs as a cloud service.

Its delicate to satisfy requirements may in some cases impose an extra layer of complexity on the on-demand (OD) ESB solution, but there are countless advantages that this can bring just because of the cloud delivery model. Mainly because of this pros and cons ongoing dispute, many enterprises chose the private cloud as the stepping-stone for a permanent migration toward such a solution.

This is most often a tradeoff decision made of keep some degree of direct control to the integration problem. For the moment, however, it turns out that setting up a private cloud is generally harder than planned. Aside from the potential impact on existing hardware and software configurations, there is the problem of managing what is proving to be a rather unwieldy new infrastructure, [13].

In any case most of the above examples provide an integration/connectivity software of some sort that enables plugging existing systems into the EBaaS and the most frequent problem immediately associated with this is the product/technology dependency. Usually this connectors include not only the cloud interface (CI) and the data interface (DI) to enable communication, but they also replicate a fair amount of the client software business logic and this makes them hard to scale and maintain. Only the above mentioned "CloudHub" implements a total of 72 connector for different software applications and it is yet far from serving at least a decent amount of market software that demands integration.

V. EBAAS REQUIREMENTS

In essence, the on-demand (OD) versus the on-premises (OP) ESB architecture have the same basic requirements, they might just be delivered in a different way. One of the things we've learned about the cloud in the last few years is that the cloud is not a product, but an architectural style or a delivery model, it's not necessarily a way to redo either the business need or the technical capabilities. Therefore, on-demand ESB will still find a need for:

- *Data aggregation*, as the on-demand EBS still needs application that can pull data from multiple sources and combine it so serve its purposes.
- *Data replication*, as getting data from A to B remains a necessity even in the cloud space.
- *Single source of truth*, as there is still a lot of value in having a single authoritative source of truth as this was the business premise of Service-Oriented Architecture as much as that may or may not have been adopted well.
- *Shared business functions*. Reusable business function must not be baked into an application or into a standalone service as that needs replication of that logic/component in every platform that integrates. In return, that should be designed as a single shared business service that everyone calls, whether they are in the cloud or on premise.
- *Distributed long-running business processes*. Not every process can be a request-response or an asynchronous message, but instead it can involve multiple systems and people. Sometime the need for this requirement is even greater when having a more distributed system.
- *Business partner integration*. As being outside the firewall, you identity management may become a little simpler, but the need is still the same even if you are delivering via a cloud technology
- *Integration software*, as the architecture still needs transformation of data, mediation, routing, orchestration and connectivity.

VI. EBAAS CONSIDERATIONS

There is a particularly strong requirement for looking into the considerations of an EBaaS vendor, but also of a client migration, or more precisely, to understand how does the hosting environment change two functionally similar products. In this perspective we find most of the time aspects that are trivial in the OP systems, maybe naturally or natively present, but they are exacerbated in the cloud.

Among the most common aspects worth mentioning, we can find:

- *Network latency*, that goes from 1 to 4 milliseconds typical latency time inside a data center to maybe 200 milliseconds while spanning continents to connect to different services; New techniques and technologies need to be used in order to ensure a responsive experience and to go around latency or accounting for it.
- *Identity management*. As and EBaaS means accessing services outside of you network, there is a delicate need to ensure that the right user is not only authenticated but also authorized to perform an operation
- *Different service-level agreements (SLAs)*. Unlike the OP case, where a service-down is easily solved by

addressing a local resource and performing a transparent operation inside an owned data center, the cloud model of the ESB assumes one vendor providing a service to dozens or hundreds of customers simultaneously; Also, there are no dedicated lines to find out information, while an EBaaS can have a different SLA that has more/less availability and all the client systems must adapt to that.

- *Data Security*. There is usually a general concern for moving private workloads to the cloud and insuring their security. In some cases it is still the customer's responsibility of encrypting data before posting it or to developing their own role management to make sure only certain people can access it. In these cases the customer is still responsible for a data-bridge even if the problem was initially caused by the provider itself.
- *Interoperability*. As you move outside your data center you don't control the endpoints anymore, you're not able to decide what protocol or product to use, you lose control over message payloads and schemas for security and transactions. You're simply connecting a lot of different products in some cases through a lot of different protocols.
- *Mobile access*, which obviously is a huge popular area right now, requires acknowledgement of its importance and effort into ensuring you are doing the right things to make it easy for people to access data in a mobile way.
- *Management*, emphasizing the question of still having to use 32 different dashboards to set up each cloud configuration or is there some kind of consolidation tool that enables managing each of this different application integrations or data platform in a centralized way.
- *Monitoring*. We develop in this situations significant concerns for data-volume and privacy monitoring, regardless of having data going from cloud to cloud or even from cloud to on premise. There is most probably the need of deploying more than one tool to monitor data traffic and usage.
- *Changing schemas*. Data is generally much less constant as you're integrating with cloud systems mostly because of the dependency linkage with someone else's upgrade cycle. If the EBaaS you're using decides the addition of a new element to a particular data schema, its arrival is eminent, regardless of the clients' approval and if there's an online solution baked in with that schema, there is the necessity of updating it before the lunch to ensure the integrations don't break. Although there is the drawback of a more volatile environment, the benefit is the early accessing of innovation.
- *Services not Servers*. Something that most people tend to overlook is that they are not able to put configuration onto individual servers or to change registry settings, but they are accessing service endpoint, that may be ephemeral. There is not as much transparency to the underlying infrastructure while dealing with a service that abstracts the server underneath.
- *Connectivity*. This is probably related tightly to SLAs, but there is also a need to deal with offline systems. Questions like the following must be taken into account: What happens when a user can't connect to a service? What happens when a data center can't connect to the cloud platform? Do things fall apart or is there a fallback mechanism that localizes the integration problem so the cracks don't spread to other places in the system?

None of the above mentioned aspects are impossible to satisfy in any way, but like mentioned before, understanding the needs and requirements of the clients before designing a and delivering an application adds great value. As much as this may seem obvious, this is not always taken into account.

VII. A FIRST GLANCE INTO THE CONNECTIVITY MODEL

Like before mentioned, there is strong lack of connectivity abstraction in most of the connector software provided by the EBS integration platforms available in the market. This connector applications usually have a Data Interface and a Cloud Interface, but most commonly, they include a significantly big block of code that embeds all the business logic of the connecting application. This introduces a great level of coupling and require upgrades of the connector with every upgrade of the connected application and the infrastructure bus.

In the ideal connectivity model, the EBaaS only uses a Remote Agent or a thin peace of software that abstracts away anything but the DI and only knows how to communicate with the synchronized application.

The entire business logic in this case together with the CI would be moved into the cloud themselves, in a so called Cloud Connector Space (CCS), and inject any information needed in the Remote Agent at sun time.

Furthermore, in case of any change in the business logic of the solution or in the cloud API, there would be no need whatsoever to touch the Remote Agent, redeploying the CCS would enable applying any software changes needed.

A. A first glance into the connectivity model

First of all, the above design should significantly decouple the connected entities, providing a clean and fast way to apply a change or a fix directly into the cloud and have it available instantly for all the clients.

Secondly, by providing the ability of absorbing and executing intelligence at runtime (a form of scripting more or less), the Remote Agent will be able to query all kind of information in real time, delivering in a centralized way profiling and statistical data gathered form all the live connection.

On top of this, abstracting the OP connectivity concept into this thin Remote Agent enables a high level of scalability. The OD EBS examples in the chapters above show significant numbers of connectors (close to one hundred) developed for every one of this systems. The EBaaS Remote Agent – Cloud Connector Space (RA-CCS) design not only makes that a lot more scalable, but also enables quick reaction to a new systems that needs plugging into the EBaaS, diminishing costs and effort significantly.

B. Stability and Data Traffic Optimization

Data Traffic Optimization represents of course one of the pillars of connectivity and communications and has always received the most careful dedication in improvement and innovation of such solution. The RA-CCS design tries to do the same.

As the RA has no knowledge whatsoever about the transferred data and it has no reason to manipulate it, a data flattening process is be applied at every communication end, enabling data compression and optimal partitioning.

Furthermore, the profiling and gathering of data make itself useful together with the RA's capacity of following the CCS's needs. In case of a traffic jam, a one-server-downtime, or even in respect to client subscription type or a client one-time-high-traffic-event, the CCS is able to determine its available resources and dynamically allocate bandwidth among live connection, reducing the less important ones and increasing the ones it needs to over-satisfy.

More than making it powerful, this design will bring stability into the product, as the cloud will never be flooded or surprised by any situations and though in some critical cases will choose to decrease performance for some customers/connections it will always be responsive to dedicated channels.

C. API / SDK

In terms of providing the clients with an API, this is generally something required for such an integration solution and of course some of the customers will have a development team assign to building a "custom" integration connector software that serves their needs. Well these developers will not have to care about pieces of communication they should not be exposed to, but most import, they will not be able to write code that brakes any of this components integration adding an extra layer of protection to the designed solution.

More than this, we globally agree that the scalability and the flexibility of a platform is something greatly valued nowadays and the universal metric for the "quality" of an framework API is the effort to solution-complexity ratio or in other words its productivity. The productivity can be increased by increasing the abstraction level of the API, but of course at the cost of the flexibility of this API so a practical equilibrium is required here.

What about customers that demand fairly simple integration cases that are not by default supported by this solution? Should they need development resources again, both human and technical, meaning loads of effort and money implicitly? Well, here comes the interesting part: on top of the code API, the solution provides also a high level SDK. Shaped as a graphical tool, this would enable companies to empower not development teams, but consultants to design a custom integration endpoint and plug in into the solution.

The SDK would provide an ETL (Extract-Transform-Load) like tool, that would allow parsing the schema of the new data format and easily map it via some intuitive GUI to the existing solution data format. Using this, the integration and plugin of a new system can be reduced from weeks or months in a classical code-writing way, depending on the development team experience and expertise with such systems, to days or even hours, depending on the SDK automation level.

D. Online vs. Offline Synchronization

If we are talking about cloud services, then we are of course talking about real-time synchronization between our different integrated systems. Let's take mobility as an example and consider that our platform could provide integration with a mobile solution that serves partial or extended functionalities from an ERP system. Whether you are accessing a dashboard from you tablet or approving a sales document on you smart phone you would like to have displayed up to date and accurate information and your mobile interaction to be reflected in the ERP as soon as possible.

Sure, this is what most of the Cloud based solution offer, but sometimes is not enough. Imagine a technician or a sales employee traveling to a costumer site that has no internet connection because of some interference proof shielding, no network coverage in the area or some random technical reason. As a business owner, I would not want him to fill out a piece of paper instead of an electronic form and wait until he reaches internet connectivity to put that in the system. Or even worst, he may miss technical or pricing documents that he did not synchronize prior to his arrival.

The proposed connectivity model would therefor enable both synchronous and asynchronous workflows, providing full capabilities to the integrated applications offline, and updating their information via the bus infrastructure as soon as they connect back.

E. Conflict Resolving

If we talked about asynchronous workflows and offline synchronization, then of course the following question to be asked is whether we will encounter conflicts in our data. The answer in this case is obviously yes. One may argue that there will be no to salesmen working on the same sales document at the same time or no two technicians filling out the details for the same service call on two different mobile clients integrated with an ERP. Nevertheless, there is always sensitive shared data prone to conflict and that needs a resolution.

Classic SOA architectures are based on the "Single Source of Truth" principle, where there's always a know-it-all entity (the ERP system in our case) to which all data conflicts are resolved against. Well, let's think this through. Maybe we have indeed two mobile clients modifying simultaneously the same data and both doing it for legit reasons. Solving the problem in an automated manner would mean either discard the information of the second client synchronizing or overwrite the data persisted by the first one. Each of these approaches would result in data corruption.

For this reason, the solution would provide integrated conflict resolving mechanisms that would allow the users to have a preview of the data differences whenever they are trying to perform a change against not up to date data, analogous with what software versioning systems do in such cases, and allow them to decide which combination of data is correct and commit it into the system.

F. Extensive Logging

Like previously mentioned, cloud application have developed sensitive concerns for monitoring data volumes and privacy in such a distributed solution. Therefore, there is a need for deploying and deploying more than one logging or monitoring system to provide accurate information to clients about their data.

Also the format of this information should be standardized and centralized, traveling from every component of the solution into a single "control center" so it can boil down into a monitoring dashboard. With such an instrument in place, if a data package is lost, the client would easily be able to track it down to the component that failed to process it know what it has to do in order to recover it.

This kind of transparency via logging is a must in a highly distributed integration solution, therefor has to receive particular attention in the connectivity model.

G. Dedicated Communication Channel

If we talk about conflict resolving and centralizing logging information from across the solution, than we implicitly have to consider that more than one package type take will travel on the Cloud bus, namely on top of pure data packages (domain objects), there will be state signals, health information or verbose logging packages.

The simplest solution for managing this requirement is to implement package priority in the bus protocol and have each component choose at their OSI (Open Systems Interconnection *ISO/IEC 7498-1) application layer the priority order of the next package to send. This works for many of the cases, but again shows some limitations.

Let's say that one of the components is transferring an attachment or a randomly large object across the communication channel. Even if the OSI transport layer brakes this properly into manageable size packages, the application layer will still hold on to that object until it finishes it, at least if it does not implement some truly complex multiplexing mechanism.

In some cases, the time until the application layer finishes that object is not acceptable as a wait time for something like a priority state information, especially if the end-to-end travel time will sum up more than one bottle neck like this. For this reason, the ideal solution for solving this requirement is a dedicated channel, independent and self-sufficient, that can serve time critical information traveling across components.

VIII. CONCLUSIONS

The proposed solution implies less than simple requirements, but if we are to consider the target market size, the numerous implementation that are constantly being developed, extended and shared all around the world and the exacerbated interest in the Cloud world, from personal use to solid enterprise solutions, we would be hypocrites not to admit the importance and potential of this research domain.

Promoting a completely-modularized, highly-decoupled, easily-scalable and maintainable implementation, the "EBaaS Ideal Connectivity Model" builds its core values around traffic optimization, client usability and a high level of transparency via intense monitoring, but also simplicity in plug-in capabilities and, therefore, adheres to what seems to be the future in Cloud integration connectivity solutions.

With regards to the present writing's perspective, this research direction aims towards identifying the key strengths and weaknesses across several connectivity frameworks in the on-premises and the on-demand worlds and processing them into an architectural overview of a simple, but solid cloud connectivity model. Serving as a pillar in tying up the loose ends of today's connectivity externalization problems, like trust, privacy, security, performance, reliability or interoperability, the EBaaS model plans to encapsulate both hardly technical and more sensitive aspects of a scalable and maintainable cloud based integration platform.

REFERENCES

[1] Mladen A. Vouk, "CloudComputing–Issues, Research and Implementations", Journal of Computing and Information Technology - CIT 16, 2008, 4, 235–246
[2] Ilango Sriram, Ali Khajeh-Hosseini, "Research Agenda in Cloud Technologies", 19 January 2010
[3] David Chappell, "Cloud Computing: The Big Picture", http://www.pluralsight.com, 2 September 2012
[4] Qi Zhang, Lu Cheng, Raouf Boutaba, "Cloud computing: state-of-the-art and research challenges", 20 April 2010 © The Brazilian Computer Society
[5] Richard Seroter, "Patterns of Cloud Integration", http://www.pluralsight.com, 4 September 2013
[6] Mohammad A. Rashid, Liaquat Hossain, Jon David Patrick, "The Evolution of ERP Systems: A Historical Perspective", Copyright © 2002, Idea Group Publishing
[7] "What Is ERP?" http://www.netsuite.com/, Copyright © 1998 - 2013 NetSuite Inc.
[8] "ERP History", http://www.saptech.8m.com/erp_history.htm, 2013
[9] Aberdeen Group Research Report, "Aging ERP: When Old ERP is Too Old", ©2012 The Resource Group. All rights reserved.
[10] "Independent, Expert and Balanced ERP Software Analysis, Reviews & Insight", http://www.erpsoftware360.com/, 2013
[11] Mohammad Kif, "Microsoft's Enterprise Service Bus (ESB) Strategy", http://blogs.msdn.com/, 23 January 2007
[12] Joe McKendrick, "Enterprise Service Busted?" July 22, 2008, http://www.zdnet.com/blog/service-oriented
[13] Arthur Cole, Venturing onto the Private Cloud, 04 JUN, 2010, http://www.itbusinessedge.com/
[14] Vance McCarthy, "Fiorano Launches Cloud-Based ESB for SOA, Integration", http://www.idevnews.com/, 2013