International Journal of Computer & Information Technologies (IJOCIT)
www.ijocit.ir
ISSN = 2345-3877

Demonstrate the Crucial Need to Review the Architecture of Distributed Systems and to investigate the Deficiencies and problems

Masoud Rafighi[1], Nasser Modiri[*2], Yaghoub Farjami[3]
Department of IT[1,2,3]
Qom University[1,3], Zanjan Azad University[2]
Qom [1,3], Zanjan[2]
Iran[1,2,3]

Masoud_r62@yahoo.com[1], nassermodiri@yahoo.com[2], farjami@gmail.com[3]

Abstract:- In this paper, the architecture of distributed systems are investigated. Three of the most important and most efficient architectures are compared and their problems will express. Every one of these architectures is unable and ineffectiveness to answer considered deficiencies for distributed systems. It is necessary to have architecture which response deficiencies and problems.

Keywords: Data-centric architecture, Pipe and filters architecture, Architecture Client/Server.

1. Introduction

Customers' requirements control the creation and deployment of software. Customers demand more and better functionality, they want it tailored to their needs, and they want it "yesterday." Very often, large shops prefer to develop their own in-house add-ons, or tweak and replace existing functions. Nobody wants to reinvent the wheel, but rather to integrate and build on existing work, by writing only the specialized code that differentiates them from their competition. Newer enterprise-class application suites consist of smaller stand-alone products that must be integrated to produce the expected higher-level functions and, at the same time, offer a consistent user experience. The ability to respond quickly to rapid changes in requirements, upgradeability, and support for integrating other vendors' components at any time all create an additional push for flexible and extensible applications.

Down in the trenches, developers must deal with complex infrastructures, tools, and code. The last thing they need is to apply more duct tape to an already complex code base, so that marketing can sell the product with a straight face.

Software Architecture [31; 32] describes the high-level structure of a system in terms of components and component interactions. In design, architecture is widely recognized as providing a beneficial separation of concerns between the gross system behavior of interacting components and that of its constituent components. Similarly this separation is also beneficial when considering deployed systems and evolution as it allows us to focus on change at the component level rather than on some finer grain.

For instance, previous work described some of the issues involved in specifying a limited form of dynamic software structure for distributed systems in which the set of components and their interaction change as execution progresses and the system evolves [33]. A change to the software architecture could occur either as the result of some computation performed by the system or as a result of some external management action such as to insert a new component and to change those connections within the system to accommodate the new component. Management actions are performed by a configuration manager [34]. Which maintains an overall view of the structure of a system in terms of components and their interconnections and performs changes in the context of that view? In essence, the configuration manager is responsible for ensuring that an executing system conforms precisely to its architectural specification. This approach can however be too restrictive for current dynamic, open systems.

The paper is organized in six sections. Section second, introduces software architecture. Distributed systems are discussed on Section three. Architectures for Development of Software Distributed are mentioned in section forth. Section fifth includes Measurement and Analysis of the Architecture Criteria. Section sixth discuss about problems. Finally, conclusions are presented in section seven.

2. Software Architecture

Architecture, the fundamental organization of a system consisting of components, each of which is associated with each other and with the system and the principles governing its design and evolution is. Software architecture is the choice of a general structure for implementing a software project based on a set of user requirements and business of software systems which it can implement our requirements, optimize the software quality, production and

maintenance and accelerate it. Nowadays due to the development of distributed systems that are constantly changing need for a flexible architecture can be felt more than ever [28].

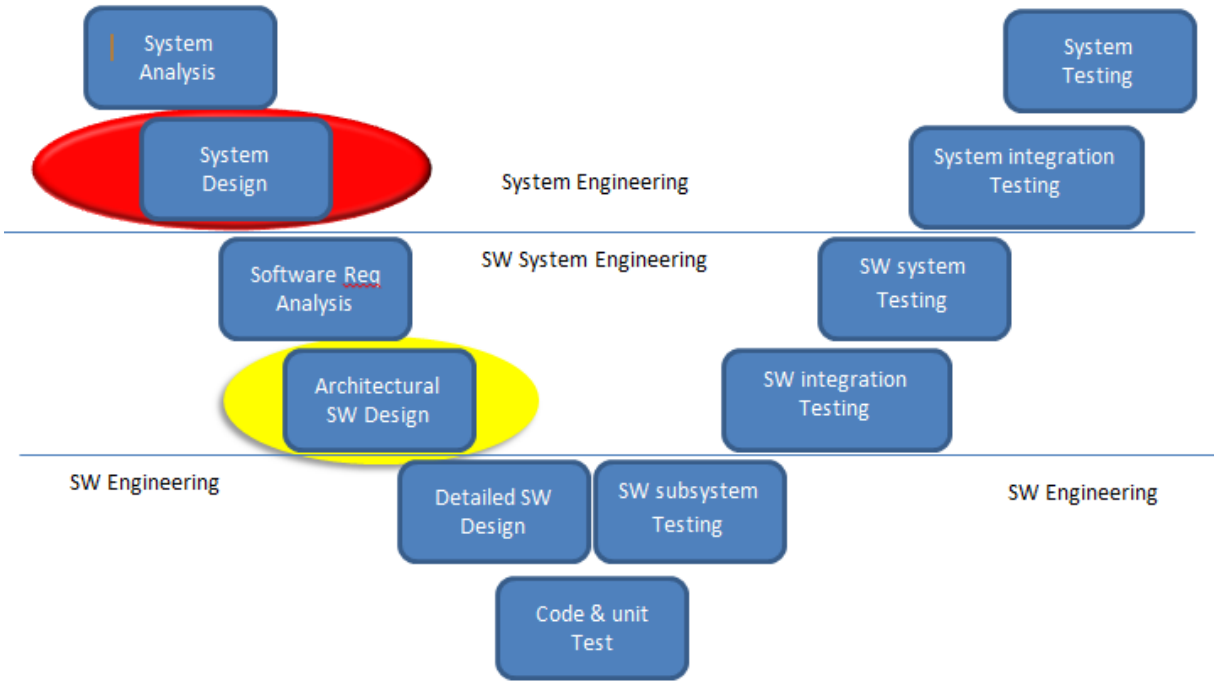

Figure 1: Architecture: Place in System Development Cycle [28]

3. Distributed Systems

A distributed system is essentially a computer system where components of the system are held on physically separated, autonomous computers. These machines communicate through the use of a computer network, either a fixed or, in the case of mobile applications, a wireless network. The distributed systems appear to users as a single, integrated computing facility.

In recent years, distributed systems have become increasingly popular and important in modern computing. They provide opportunities for increasing the reliability, availability and performance of applications. However, perhaps the most important feature of a distributed system is that it allows the integration of existing systems. Companies do not wish to rewrite large numbers of legacy applications and a distributed system allows these applications to be integrated in a relatively straightforward manner.

A distributed system may comprise components written in a number of different programming languages, running on different operating systems on a variety of computer architectures.

In many cases, a distributed system may be cheaper than a single, centralized system. A large number of small, low-power systems may prove

cheaper to purchase than a single mainframe or supercomputer. This is the approach employed in Beowulf clusters, which allow a collection of computers to act as a single large computer.

There are obviously many significant disadvantages to distributed systems. They are much more complicated to design, build and maintain than an equivalent centralized system. There are a large number of possible failures that could occur in a distributed system, far more than would be found in a centralized system. Because of this, a distributed system will have multiple points of failure, increasing the likelihood of the system not functioning correctly. Communication over a network will always be far slower and less reliable than communication over a local bus, which has a significant effect on the performance of a distributed system [4, 5, and 29].

Distributed systems architectures

- Client--server architectures
  - ✓ Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services.
- Distributed object architectures
  - ✓ No distinction between clients and servers. Any object on the system may provide and use services from other objects [30].

4. Architectures for Development of Software Distributed

4.1. Data-Centric Architecture

The goal of this architecture is to maintain the integration and the ability of Aggregation. The word "data-centric" refers to systems that the availability and timeliness of the data is an appropriate descriptive of system performance. A client runs on a set of independent control field and common data that is accessed by all customers and it can be as a passive source (such as a file) or an active source (Blackboard).

The concept of association can refer to two groups:

- Common data act as a passive source (such as a file)
- Common data act as an active source (such as a blackboard)
- The blackboard against passive source sends massage to customers on the time of changing data so it is active. Blackboard with this style, as it would include arrows that can be derived from the shared data. The architectural style is always expanding and improving importance. This is due to a structural solution is to achieve integration capabilities In many systems, especially systems of pre-built components, data integrity provided by mechanism blackboard. In this style, a major advantage is that customers are

available as independent and common data independent of the customer. Therefore, this style is scalable and can easily add new customers.

This style has high Corrigibility too and it's Due to the change of each customer is having no effects on other customers. In this style, if a connection is established between the customers In spite of the fact that it will reduce Corrigibility, it increases the efficiency [16, 17, and 18].

4.2. Pipe and Filters Architecture

Pipe and Filter emphasize on gradual conversion and processing data with consecutive components. This architecture is a popular style of UNIX operating system family. In this style Filters are components and data will conversion gradually. The pipes are connectors which don't get any state, they just used between filters for moving. The rules that have governed in this style show how close the pipes and filters are also specified. Every pipe has one source end which is connect to output port and one sink end which is connect to input port [23].

4.3. Architecture Client/Server

This architecture used the server and client with different characteristics. The server will do heavy processes and the client will do light processes. Client and server by sending request and response show the aspects of cooperation in the processing operations. In this architecture, the operating which does on a program devise between the server and the client [9, 10].

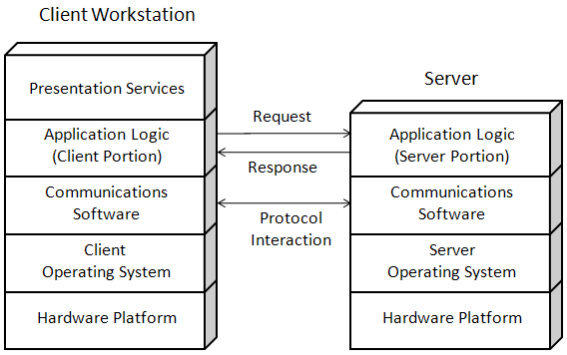

Figure 2: Server & Client Arcgitecture

5. Measurement and Analysis of the Architecture Criteria

5.1. Layout of components

Components as the original block and computational entities participating in the construction of system throw internal computation and external communication do their choruses. Every component communicates with environment by one or more port. A user interface can be a common variable; the name of a procedure which calls from other component; it is a set of events that can occur as a component and other mechanisms. Properties of a component, the data for analysis and software implementation specifies.

5.2. Create

Configuration is a connected graph which sometimes referred to as the topology which is composed of components and connectors and describes the structure of architecture.

5.3. Connection

When connector makes a connection between two components, component defines an interface. And every component can have several interfaces. An interface is concerned to just one component and every interface of one component can connect to several interfaces in other components. For example in Bus-Oriented architecture the interface of every component is connected to the bus connector and so it will connected to several interface in other components. Attributes can also be indicated by some of the feature, such as communication, buffering capacity and so on.

5.4. Development

Develop and promote will be Causes the development and software update in computer systems so an important metric that can be considered in the selection of the architecture is extensible metric. The software architecture must be Extensibility. We evaluate it since this metric is a major role in architecture.

5.5. The Main Advantage

Each of software architecture has advantages compared to other architectures. The software architecture eliminates defects in other architectures and complement previous architectures.

5.6. The Main Problem

Although each software architectures try to be the best and perfect, but with the development of information systems and their development is still facing problems and In some cases, the problems faced.

These criteria were chosen only for the problems and shortcomings of Distributed software development architectures and of course there are other factors and criteria that are not effective in this research. To see a full description and explanation of software metrics can be [M. Shaw and D. Garlan, 1996] presented [27].

Table1 : Compare Architectures

Architecture / Criterion	Data- centric	Pipe and filters	Client/server
layout	Data is stored in a database and a common data is	Filters are component and data will conversion gradually	Data is exchanged between server and client [22].

	accessed by all customers [24,25].	[12,14].	
Creation	The architecture emphasizes the accessing and updating data [24,26].	The architecture emphasizes the gradual conversion and processing data with consecutive components [11,15].	This architecture used the server and client and the operating which does on a program devise between the server and the client [19,20].
Connection	The connection is done in two ways: When the share data as a passive source acts like file. When the share data is as a blackboard [24,26].	The pipes are connectors which don't get any state, they just used between filters for moving [12,11].	Connect the customer / client to the server by using the network platform [21].
development	**It** has high Corrigibility Due to the change of each customer is having no effects on other customers [24,26].	For development every pipe has one source end which is connect to output port and one sink end which is connect to input port [15].	Development to some extent that the server and network can hold it. So in fact system development is limited [19,20].
Elected or a combination of other architectures	it's not selected from another architecture [26].	This architecture is a popular style of UNIX operating system family [14,11].	it's not selected from another architecture [19,20].
The main advantage	The data integrity provided by mechanism blackboard and common data independent of the customer. Therefore, this style is scalable and can easily add	There is the possibility of balance and distribution of each filter can be alone and isolated from the rest of itself process. Capability that allows modifying and re-use of the system as a simple combination of individual components	Exploiting the potential of existing hardware, according to the principle of division operations Optimizing the use and the use of shared resources Optimizing the ability of users of the different

	new customers [24,25].	behavior with the filter [12,15].	activities [19,20].
The main problem	In this style if a connection is established between the customers In spite of the fact that it will reduce Corrigibility, it will increase the efficiency [24,26].	arrange filter is difficult In addition, the filters can't no way to solve the problem together therefore, efficiency will reduce [14,15]	No encapsulates the strategic policies Software Reducing the efficiency of the program by raising the number of concurrent users Improve application performance and reform has been seriously challenged [22].

6. Problems Discussion

The feasibility survey was conducted for exploring attitudes of the users and potential customers. It showed that main obstacles which hinder usage of service are related to possible cloud service termination or failure and vendor lock-in [1]. The rule engine component enables to inform the customer. If he can retrieve the data bach from cloud in the required format and ensures possibility to use the backup data with the local system of the customer and prevent from vendor lock-in situation[1]. Availability, data lock-in, data confidentiality and auditability are the obstacles which affect adoption of cloud computing [2]. Although cloud computing providers are facing several architecture and design challenges, however, security concerns, interoperability, data lock-in are on top of those challenges. Most of the clouds are vendor-locked, as several cloud providers offer APIs (application programming interfaces) that are well-documented, but are mainly proprietary and exclusive to their implementation and thus not interoperable[3].

For 20% of the respondents, risk of vendor lock in, loss of control, and security were sources of concern. The ability to meet government and industry standards was not seen as a concern, as none of the respondents selected this option.[6] Now, certain characteristics of this alternative make it attractive for SMEs: greater adaptability, no vendor lock-in, property of the source code, and cost comparable to other alternatives [7]. This last problem has been further pursued by IS researchers who have looked at package customization and organizational adaptation as alternative ways of resolving such misalignment [7].

At present, there are many companies implement Enterprise Resource Planning (ERP), some companies choose to buy the ERP software directly, or hire the professional group coding software for the companies. However, due to the poor flexibility of the system, and not very appropriate for business processes and management concepts, Some companies hitch have lots of profits choose to self-development the ERP system [8]. ERP system change the business process of the enterprises, and it is difficult to personnel adapt to the new system, as a result, it will also prolong the whole time in ERP implementation [8]. In this condition, the system can better focus on needs of users. How to solve these business problems and technical details will be completed through the conversion tool. Although the definition of the conversion is difficult, when business needs changes, it can be used again. In the long run, this effort has positive effect to the rapid development [8].

By analyzing the existing system and the resources in the world have pointed to the problems, problems that are not responsive architectures are as follows:

- Extensibility problem involving (the laws have changed, change in data, the changes in the organization, integration, change in operations, changes in systems, developing new systems).
- Problem of imprisonment or trapped data.
- Tele-programming problem, the only programmer can develop the system further.
- To solve the above problems, there are solutions which are listed below:

One effective way to make your application extensible is to expose its internals as a scripting language and write all the top level stuff in that language. This makes it quite modifiable and practically future proof (if your primitives are well chosen and implemented). A success story of this kind of thing is Emacs. I prefer this to the eclipse style plugin system because if I want to extend functionality, I don't have to learn the API and write/compile a separate plugin. I can write a 3 line snippet in the current buffer itself, evaluate it and use it. Very smooth learning curve and very pleasing results.

One application which I've extended a little is Trace. It has a component architecture which in this situation means that tasks are delegated to modules that advertise extension points. You can then implement other components which would fit into these points and change the flow.

But due to the distributed systems need database, these solutions can’t be hopeful way. Like most things in life, taking the time to plan ahead when building a web service can help in the long run

understanding some of the considerations and tradeoffs behind big websites can result in smarter decisions at the creation of smaller web sites. Below are some of the key principles that influence the design of large-scale web systems:

- **Availability:** The uptime of a website is absolutely critical to the reputation and functionality of many companies. For some of the larger online retail sites, being unavailable for even minutes can result in thousands or millions of dollars in lost revenue, so designing their systems to be constantly available and resilient to failure is both a fundamental business and a technology requirement. High availability in distributed systems requires the careful consideration of redundancy for key components, rapid recovery in the event of partial system failures, and graceful degradation when problems occur.
- **Performance:** Website performance has become an important consideration for most sites. The speed of a website affects usage and user satisfaction, as well as search engine rankings, a factor that directly correlates to revenue and retention. As a result, creating a system that is optimized for fast responses and low latency is the key.
- **Reliability:** A system needs to be reliable, such that a request for data will consistently return the same data. In the event the data changes or is updated, then that same request should return the new data. Users need to know that if something is written to the system, or stored, it will persist and can be relied on to be in place for future retrieval.
- **Scalability:** When it comes to any large distributed system, size is just one aspect of scale that needs to be considered. Just as important is the effort required to increase capacity to handle greater amounts of load, commonly referred to as the scalability of the system. Scalability can refer to many different parameters of the system: how much additional traffic can it handle, how easy is it to add more storage capacity, or even how many more transactions can be processed.
- **Manageability:** Designing a system that is easy to operate is another important consideration. The manageability of the system equates to the scalability of operations: maintenance and updates. Things to consider for manageability are the ease of diagnosing and understanding problems when they occur, ease of making updates or modifications, and how simple the system is to operate. (I.e., does it routinely operate without failure or exceptions?)
- **Cost:** Cost is an important factor. This obviously can include hardware and software costs, but it is also important to consider other facets needed to deploy and maintain the

system. The amount of developer time the system takes to build, the amount of operational effort required to run the system, and even the amount of training required should all be considered. Cost is the total cost of ownership.

Each of these principles provides the basis for decisions in designing distributed web architecture. However, they also can be at odds with one another, such that achieving one objective comes at the cost of another. A basic example: choosing to address capacity by simply adding more servers (scalability) can come at the price of manageability (you have to operate an additional server) and cost (the price of the servers).

When designing any sort of web application it is important to consider these key principles, even if it is to acknowledge that a design may sacrifice one or more of them.

When it comes to system architecture there are a few things to consider: what are the right pieces, how these pieces fit together, and what the right tradeoffs are. Investing in scaling before it is needed is generally not a smart business proposition; however, some forethought into the design can save substantial time and resources in the future.

This section is focused on some of the core factors that are central to almost all large web applications: services, redundancy, partitions, and handling failure. Each of these factors involves choices and compromises, particularly in the context of the principles described in the previous section.

When considering scalable system design, it helps to decouple functionality and think about each part of the system as its own service with a clearly defined interface. In practice, systems designed in this way are said to have a Service-Oriented Architecture (SOA). For these types of systems, each service has its own distinct functional context, and interaction with anything outside of that context takes place through an abstract interface, typically the public-facing API of another service.

Deconstructing a system into a set of complementary services decouples the operation of those pieces from one another. This abstraction helps establish clear relationships between the service, its underlying environment, and the consumers of that service. Creating these clear delineations can help isolate problems, but also allows each piece to scale independently of one another. This sort of service-oriented

design for systems is very similar to object-oriented design for programming.

Another key part of service redundancy is creating a shared-nothing architecture. With this architecture, each node is able to operate independently of one another and there is no central "brain" managing state or coordinating activities for the other nodes. This helps a lot with scalability since new nodes can be added without special conditions or knowledge. However, and most importantly, there is no single point of failure in these systems, so they are much more resilient to failure.

As they grow, there are two main challenges: scaling access to the app server and to the database. In a highly scalable application design, the app (or web) server is typically minimized and often embodies a shared-nothing architecture. This makes the app server layer of the system horizontally scalable. As a result of this design, the heavy lifting is pushed down the stack to the database server and supporting services; it's at this layer where the real scaling and performance challenges come into play.

Finally, another critical piece of any distributed system is a load balancer. Load balancers are a principal part of any architecture, as their role is to distribute load across a set of nodes responsible for servicing requests. This allows multiple nodes to transparently service the same function in a system. Their main purpose is to handle a lot of simultaneous connections and route those connections to one of the request nodes, allowing the system to scale to service more requests by just adding nodes. Load balancers are an easy way to allow you to expand system capacity, and like the other techniques in this article, play an essential role in distributed system architecture. Load balancers also provide the critical function of being able to test the health of a node, such that if a node is unresponsive or over-loaded, it can be removed from the pool handling requests, taking advantage of the redundancy of different nodes in your system

7. Conclusion

The comparison of these methods with parameters (layout, create, connect, development, main advantage, the main problem) to the conclusion that although each of these architectures claim that are suitable for distributed system or changing, but in practice they aren't responsive these changes in system.

A repeating theme in my development work has been the use of or creation of an in-house plug-in architecture. I've seen it approached many ways - configuration files (XML, .conf, and so on),

inheritance frameworks, database information, libraries, and others. In my experience:

•A database isn't a great place to store your configuration information, especially co-mingled with data

•Attempting this with an inheritance hierarchy requires knowledge about the plug-ins to be coded in, meaning the plug-in architecture isn't all that dynamic

•Configuration files work well for providing simple information, but can't handle more complex behaviors

•Libraries seem to work well, but the one-way dependencies have to be carefully created

These examples seem to play to various language strengths. Is good plugin architecture necessarily tied to the language? Is it best to use tools to create plugin architecture, or to do it on one's own following models?

Can you say why plugin architecture has been a common theme? What problems does it solve, or goals does it address? Many extensible systems/applications use plugins of some form, but the form varies considerably depending upon the problem being solved.

That's a great question. I'd say there are some common goals in an extensible system. Perhaps the goals are all that's common, and the best solution varies depending on how extensible the system needs to be, what language the system is written in, and so on. However, I see patterns like IOC being applied across many languages, and I've been asked to do similar plugins (for drop in pieces responding to the same functionality requests) over and over again. I think it's good to get a general idea of best practices for various types of plugins

In a world of increasingly complex computing requirements, we as software developers are continually searching for that ultimate, universal architecture that allows us to productively develop high-quality applications. This quest has led to the adoption of many new abstractions and tools. Some of the most promising recent developments are the new pure plug-in architectures.

Mentioned pathologic problems still remain open in systems and not with these architectures resolve common problems and the need for a revision in architectural theory.

References

[1] Dalia Kriksciuniene, Donatas Mazeika, (2011), "Cloud Computing and the Enterprise Needs for Data Freedom, The Third International Conference on Future Computational Technologies and Applications," FUTURE COMPUTING .

[2] Dr. Ashu Khanna, Dr. Vivek Kumar and Indu Saini, (2012), "ERP SYSTEMS: PROBLEMS AND SOLUTION WITH SPECIAL ,REFERENCE TO SMALL & MEDIUM ENTERPRISES," International Journal of Research in IT & Management," IJRIM , Volume 2,no. 2 .

[3] Ahmed Elragal , Moutaz Haddara, (2012), "The Future of ERP Systems: look backward before moving forward, CENTERIS, prise Information Systems / HCIST, International Conference on Health and Social Care Information Systems and Technologies, ELSEVIER, Procedia Technology, pp. 21 – 30.

[4] R. Allen, (1997), "A Formal Approach to Software Architecture," PhD Dissertation, Carnegie Mellon University.

[5] D. Garlan, D. Perry, (1995), "Introduction to the Special Issue on Software Architecture. IEEE Transactions on software Engineering," Vol. 21, No.4, p. 269-274.

[6] Jacek Lewandowski, Adekemi O. Salako, Alexeis Garcia-Perez, (2013), "SaaS Enterprise Resource Planning Systems: Challenges of their adoption in SMEs, IEEE 10th International Conference on e-Business Engineering,

[7] Placide Poba-Nzaou & Louis Raymond, (2013), "Custom Development as an Alternative for ERP Adoption by SMEs: An Interpretive Case Study, Information Systems Management," 02 Sep 2013.Published online.

[8] Liu Chen, Liu Xinliang, (2012), "Self-development ERP System Implementation Success Rate Factors Analysis," IEEE XPLORE Symposium on Robotics and Applications(ISRA) .

[9] Benatallah, B.; Casati, F.; Toumani, F. (2004), "Web service conversation modeling: A cornerstone for e-business automation". *IEEE Internet Computing* **8**: 46.

[10] Tolia, Niraj; Andersen, David G.; Satyanarayanan, M. (2006), "Quantifying Interactive User Experience on Thin Clients" (PDF). *Computer* (IEEE Computer Society) **39** (3).

[11] Gokhale, S.S. , Yacoub, S. (2005), "Performability analysis of a pipeline software architecture" Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International,pp. 77 - 82 Vol. 2, pp. 26-28.

[12] Ernst-Erich Doberkat, (2003), "Pipelines: Modelling a software architecture through relations", Acta Informatica, September 2003, Volume 40, No. 1, pp. 37-79

[13] Manuel Oriol, Thomas Gamer, Thijmen de Gooijer, Michael Wahler, Ettore and Ferranti, (2013), "Fault-tolerant fault tolerance for component-based automation systems, in: Proceedings of the 4th International ACM SIGSOFT Symposium on Architecting Critical Systems (ISARCS 2013)," Vancouver, Canada.

[14]http://www.cs.sjsu.edu/~pearce/modules/patterns/distArch/pipeline.htm [Accessed on 1 September 2015]

[15] Mary Shaw, (1996), "Some Patterns for Software Architectures" Pattern Languages of Program Design, Vol. 2, pp. 255-269, Addison-Wesley..

[16] G. F. Ciocarlie, H. Schubert and R. Wahlin, (2010), "A Data-Centric Approach for Modular Assurance" Real-Time Innovations, Inc.

[17] G. Carneiro, N. Vasconcelos**, (2005), "**A database centric view of semantic image annotation and retrieval**"**SIGIR '05 Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval Pages 559-566.

[18] R.Chetan, D.V. Ashoka, (2012), **"**Data mining based network intrusion detection system: A database centric approach" International Conference on Computer Communication and Informatics (ICCCI), ,pp. 1 – 6.

[19] J. Nieh, N. Novik, S. Yang, (2013), "A Comparison of Thin-Client Computing Architectures" , *Technical Report CUCS-022-00* (New York: Network Computing Laboratory, Columbia University).

[20] Barros, A. P.; Dumas, M. (2006). "The Rise of Web Service Ecosystems". *IT Professional* **8** (5): 31

[21] Yongsheng, H.; Xiaoyu, T.; Zhongbin, T. (2013). "An Optimization Model for the Interconnection among Peers of the P2P Network". *Journal of Applied Sciences* **13** (5): 700

[22] Dustdar, S.; Schreiner, W. (2005), "A survey on web services composition". *International Journal of Web and Grid Services* 1: 1. (2005).

[23] http://www.fullchipdesign.com/pipeline_space_time_architecture.htm [Accessed on September 2015)

[24] Lind P, Alm M, (2006), "A database-centric virtual chemistry system", *J Chem Inf Model, Vol.* 46, No.3, pp. 1034.

[25] http://www.boic.com/dbgrid.htm,jun [Accessed on Jun 2014]

[26]Wagner O. de Morais, Jens Lundström, Nicholas Wickström, "A Database-Centric Architecture for Home-Based Health Monitoring" Ambient Assisted Living and Active Aging Volume 8277 of the series Lecture Notes in Computer Science, pp. 26-34.

[27] M. Shaw, D. Garlan, (1996), "Software architecture: perspectives on an emerging discipline," Prentice Hall..

[28] M.J. Charistensen, R.H. Thayer, (2002), "The Project Manager's Guide to Software Engineering's Best Practices." Wiley.

[29] Robert Nunn, "Distributed Software Architectures Using Middleware," 3C05 Coursework 2

[30] Distributed Systems Architectures, "Based on Software Engineering," 7 th Edition by Ian Somerville.

[31] D. E. Perry, A. L. Wolf, (1992), "Foundations for the Study of Software Architectures, ACM SIGSOFT Software Engineering Notes," Vol. 17, No. 4, pp. 40-52.
[32] M. Shaw, D. Garlan, (1996), "Software Architecture: Perspectives on an Emerging Discipline," Prentice Hall.

Authors Profile

Masoud rafighi was born in Tehran, Iran on 1983/08/10. He is PHD student of Qom University. He receives M.Sc degree in computer engineering software from Azad University North Tehran Branch, Tehran, IRAN. He has recently been active in software engineering and has developed and taught various software related courses for the Institute and university for Advanced Technology, the University of Iran. His research interests are in software measurement, software complexity, requirement engineering, maintenance software, software security and formal methods of software development. He has written a book on software complexity engineering and published many papers.

Nasser Modiri received the MS degree in MicroElectronics from university of Southampton, UK in 1986. He received PHD degree in Computer Networks from Sussex university of UK in 1989. He is a lecture at department of computer engineering at Islamic Azad University of Zanjan, Iran. His research interests include Network Operation Centres, Framework for Securing Networks, Virtual Organizations, RFID, Product Life Cycle Development and Framework For Securing Networks.

Yaghoub Farjami is one of the best professors of IT department of Qom University.