ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

A Comparative Study on Architectural Styles from the Perspective of Network-Based Hypermedia Systems

A. D. Devangavi and Dr. Rohit P.
Associate Professor, Department of ISE, Basaveshwar Engineering College, Bagalkot Karnataka, India
Professor, Department of Computer Science and Information Technology, AISECT University, Bhopal, Madhya Pradesh.

Abstract- The purpose of building software is to create a system that meets or exceeds the application needs. The architectural styles chosen for a system's design must conform to its specific needs. Therefore, in order to provide useful design guidance, a classification of architectural styles should be based on the architectural properties induced by those styles. An attempt is made to present a survey of common architectural styles for network-based application software within a classification framework that evaluates each style according to the architectural properties it would induce if applied to architecture for a prototypical network-based hypermedia system. The goal is to provide a framework by which other styles can be added to the classification as they are developed

.

***Index Terms*-Framework for architecture designs, Software architecture, software architectural style, network-based application.**

I. INTRODUCTION

With the advance of reliable network technology, software development has progressed from traditional, platform-centric software construction to network-centric software evolution. An evidence of this change is largely reflected in the technologies that are supporting the emerging theory of Network-Centric Operations. Amongst these technologies is software architecture as a software engineering sub-discipline. The state-of-the-practice is characterized by differing interpretations about how we should design and implement this class of systems. We have discussed the characteristics of network based applications and the framework for the architecture for network based applications architectures. A survey of architectural styles cannot cover every type of software in existence, nor is there an incentive for doing so. The primary value of a survey such as this one is to provide design guidance during the creation of an architecture for a specific type of problem. Restricting the scope of the survey reduces the variability of styles to a set that can be reasonably compared against a system's requirements. Ideally, there would be a survey available for each particular problem, but that is impractical given the range of software applications. Styles are a mechanism for categorizing architectures and for defining their common characteristics .This survey restricts itself to the domain of network-based application architectures. Chapter 2 discusses the characteristics of network centric software systems and the framework for architectures for network based applications architectures. Various architecture styles for network based applications are compared in chapter 3.

II. CHARACTERISTICS OF NETWORK CENTRIC SOFTWARE SYSTEMS

Network-centric software systems are multifunctional systems that exhibit unique operational characteristics. A network-centric software system has: i) An underlying networked configuration that embodies the runtime environment on which the system's components interact and limits components' interaction to information exchange. ii)An emergent, dynamic runtime behavior, which means that the system's actual interacting components are not necessarily known until runtime and that the overall functionality of the system emerges from the collaborative behaviors of the components.iii)A fluid, dynamically-defined control, which means that control over the system functionality is not necessarily owned by a particular component; rather, this control changes based on which function the system is performing and which component has initiated the system's execution. This control can be either strategic or tactical control.

In addition to their operational characteristics, network-centric software systems have distinctive structural characteristics. We have identified some of these characteristics. Network-centric software systems are: i) Multi-functional systems that have a "system of systems" perspective. The components of these systems are in and of themselves large systems that work together towards the same goal. ii) Generally loosely-coupled systems.

Systems that have underlying networked architecture where data and applications are deployed on the network. Traditionally, advances in network technologies have allowed access to data in geographically remote data repositories. With network-centric computing, the focus is on providing efficient ways to access remotely-located applications. In order to be able to build software architectures for this class of systems, an architect will need to employ several architectural styles [1][13]. The following Framework gives an overview of the network-centric architectural style.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

Elements	Components type: Independent, active software nodes that have data exchange and task invocation ports (interfaces) Connectors type: Data exchange and task invocation links, which facilitates communication among the nodes.
Relations	Attachment relation associates a node's port with another node port. Port compatibility must be enforced.
Computational Model	Nodes are configured dynamically to collaboratively carry out network-centric tasks. A task can be initiated by any node in the system.
Communication Model	Nodes are connected by means of links between them. A node can communicate with multiple nodes via proper port-link combinations.
Elements Properties	Node: Name: should suggest its functionality Type: defines general functionalities and services of the node Ports: list the types of ports the node has Port Properties: depend on the type of the network communication Link: Name: should suggest the nature of the interactions it supports Type: Data exchange or task invocation link Parameters: defines the nature of the interaction and the required parameters to carry out that interaction Other Properties: depending on network communication type, it may include protocol of interaction and performance values
Topology	There is no topological model to this style. Nodes connect and disconnect to other nodes at will. The system has a "changing" topology that is unrestricted and that is based on the task being carried out.

III. ARCHITECTURAL STYLES FOR NETWORK BASED APPLICATIONS

Quality is defined as a set of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Simplicity, Visibility, Performance, functionality, Reliability, Usability, Efficiency, Maintainability, Evolvability, Portability and etc are some of the qualities that can be used to compare architectural styles. Various architecture styles and their advantages and disadvantages are as below [1]:

A. PIPE NAD FILTER (PF)

In a pipe and filter style, each component (filter) reads streams of data on its inputs and produces streams of data on its outputs, usually while applying a transformation to the input streams and computing incrementally so that output begins before the input is completely consumed [5]. The constraint is that a filter must be completely independent of other filters (zero coupling): it must not share state, control thread, or identity with the other filters on its upstream and downstream interfaces .The advantageous properties of the pipe and filter style as follows. First, PF style exhibits simplicity. Second, PF supports reuse Third, PF systems can be easily maintained and enhanced: Fourth, they permit certain kinds of specialized analysis (verifiability), such as throughput and deadlock analysis. Finally, they naturally support concurrent execution (user-perceived performance).Disadvantages of the PF style include: propagation delay is added through long pipelines, batch sequential processing occurs if a filter cannot incrementally process its inputs, and no interactivity is allowed. The style is non-interactive .These properties decrease user-perceived performance.

1. UNIFORM PIPE AND FILTER (UPF)

The uniform pipe and filter style adds the constraint that all filters must have the same interface. Restricting the interface allows independently developed filters to be arranged at will to form new applications. It also simplifies the task of understanding how a given filter works. A disadvantage of the uniform interface is that it may reduce network performance if the data needs to be converted to or from its natural format.

B. CLIENT-SERVER (CS)

The client-server style is the most frequently encountered of the architectural styles for network-based applications. A server component, offering a set of services, listens for requests upon those services. A client component, desiring that a service be performed, sends a request to the server via a connector. The server either rejects or performs the request and sends a response back to the client. Separation of concerns is the principle behind the client-server constraints. A proper separation of functionality should simplify the server component in order to improve scalability.

1. LAYERED SYSTEM (LS) AND LAYERED -CLIENT-SERVER (LCS)

A layered system is organized hierarchically, each layer providing services to the layer above it and using services of the layer below it [5]. Layered systems reduce coupling across multiple layers by hiding the inner layers from all except the adjacent outer layer, thus improving evolvability and reusability. The primary disadvantage of layered systems is that they add overhead and latency to the processing of data, reducing user-perceived performance [10]. Layered-client-server adds proxy and gateway components to the client-server style. A proxy [8] acts as a shared server for one or more client components, taking requests and forwarding them, with possible translation, to server components. LCS is a solution to managing identity in a large scale distributed system.

2. CLIENT -STATELESS-SERVER (CSS)

The client-stateless-server style derives from client-server with the additional constraint of no session state allowed on the server component. Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Application state is kept entirely on the client.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

These constraints improve the properties of visibility, reliability, and scalability [3]. The disadvantage of client-stateless-server is that it may decrease network performance.

3. CLIENT-CACHE-STATELESS-SERVER (C$SS)

The client-cache-stateless-server style derives from the client-stateless-server and cache styles via the addition of cache components. A cache acts as a mediator between client and server .The advantage of adding cache components is that they have the potential to partially or completely eliminate some interactions, improving efficiency and user-perceived performance.

4. LAYERED-CLIENT-CACHE-STATELESS-SERVER (LC$SS)

The layered-client-cache-stateless-server style derives from both layered-client-server and client-cache-stateless-server through the addition of proxy and/or gateway components. The advantages and disadvantages of LC$SS are just a combination of those for LCS and C$SS.

5. REMOTE SESSION (RS)

The remote session style is a variant of client-server that attempts to minimize the complexity, or maximize the reuse, of the client components rather than the server component. Each client initiates a session on server and then invokes a series of services on the server, finally exiting the session. Application state is kept entirely on the server. The advantages of the remote session style are that it is easier to centrally maintain the interface at the server, reducing concerns about inconsistencies in deployed clients when functionality is extended, and improves efficiency. The disadvantages are that it reduces scalability of the server, due and reduces visibility of interactions.

6. REMOTE DATA ACCESS (RDA)

The remote data access style [7] is a variant of client-server that spreads the application state across both client and server. A client sends a database query in a standard format, such as SQL, to a remote server. The server allocates a workspace and performs the query, which may result in a very large data set. The client can then make further operations upon the result set (such as table joins) or retrieve the result one piece at a time. The client must know about the data structure of the service to build structure dependent queries. The advantage of remote data access is improved efficiency, and visibility. The disadvantages are lack of simplicity and decreased scalability. Reliability also suffers, since partial failure can leave the workspace in an unknown state.

C. MOBILE CODE

Mobile code styles use mobility in order to dynamically change the distance between the processing and source of data or destination of results. Introducing the concept of location makes it possible to model the cost of an interaction between components at the design level. In the mobile code styles, a data element is dynamically transformed into a component.

1. VIRTUAL MACHINE (VM)

Underlying all of the mobile code styles is the notion of a virtual machine, or interpreter, style [5]. The code must be executed in some fashion, preferably within a controlled environment to satisfy security and reliability concerns, which is exactly what the virtual machine style provides. The primary benefits are the separation between instruction and implementation on a particular platform (portability) and ease of extensibility. Disadvantages of this style are reduced visibility and simplicity is reduced.

2. REMOTE EVALUATION (REV)

In the remote evaluation style [9], derived from the client-server and virtual machine styles, a client component has the know-how necessary to perform a service, but lacks the resources .Consequently, the client sends the know-how to a server component at the remote site, which in turn executes the code using the resources available there. The results of that execution are then sent back to the client. The advantages of remote evaluation include improved extensibility and customizability, and better efficiency .Simplicity and scalability are reduced .The most significant limitation, however, is the lack of visibility.

3. CODE ON DEMAND (COD)

In the code-on-demand style [9], a client component has access to a set of resources, but not the know-how on how to process them. It sends a request to a remote server for the code representing that know-how, receives that code, and executes it locally. The advantages of code-on-demand include improved extensibility and configurability, and better user perceived performance and efficiency Simplicity is reduced .Scalability of the server is improved. Lack of visibility leads to obvious deployment problems if the client cannot trust the servers.

4.LAYERED-CODE-ON-DEMAND-CLIENT-CACHE-STATELESS- SERVER (LCODC$SS)

As an example of how some architecture are complementary, consider the addition of code-on-demand to the layered-client-cache-stateless-server style discussed above. Since the code can be treated as just another data element, this does not interfere with the advantages of the LC$SS style. The advantages and disadvantages of LCODC$SS are just a combination of those for COD and LC$SS.

5. MOBILE AGENT (MA)

In the mobile agent style [9], an entire computational component is moved to a remote site, along with its state, the code it needs, and possibly some data required to perform the task. This can be considered a derivation of the remote

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

evaluation and code-on-demand styles, since the mobility works both ways. The primary advantage of the mobile agent style, beyond those already described for REV and COD, is that there is greater dynamism in the selection of when to move the code.

D. REPLICATION

1. REPLICATED REPOSITORY (RR)

Systems based on the replicated repository style [6] improve the accessibility of data and scalability of services by having more than one process provide the same service. These decentralized servers interact to provide clients the illusion that there is just one, centralized service. Improved user-perceived performance is the primary advantage, both by reducing the latency of normal requests and enabling disconnected operation in the face of primary server failure or intentional roaming off the network. Simplicity remains neutral and maintaining consistency is the primary concern.

2. Cache ($)

A variant of replicated repository is found in the cache style: replication of the result of an individual request such that it may be reused by later requests. This form of replication is most often found in cases where the potential data set far exceeds the capacity of any one client. The cache style becomes network-based when it is combined with a client-stateless-server style.

E. EVENT-BASED INTEGRATION (EBI)

The event-based integration style, also known as the implicit invocation or event system style, reduces coupling between components by removing the need for identity on the connector interface. Instead of invoking another component directly, a component can announce (or broadcast) one or more events. Other components in a system can register interest in that type of event and, when the event is announced, the system itself invokes all of the registered components [10]. The event-based integration style provides strong support for extensibility and for evolution .However, like pipe-and-filter systems, there needs to be an "invisible hand" that places components on the event interface. Other disadvantages of EBI systems are that it can be hard to anticipate what will happen in response to an action (poor understandability) .Also, there is no support for recovery from partial failure.

F. OTHER HYBRID STYLES

1. C2

The C2 architectural style [1] [2] is directed at supporting large grain reuse and flexible composition of system components by enforcing substrate independence. It does so by combining event-based integration with layered-client-server. Asynchronous notification messages going down, and asynchronous request messages going up, are the sole means of intercomponent communication. This enforces loose coupling of dependency on higher layers and zero coupling with lower levels, improving control over the system without losing most of the advantages of EBI. The introduction of layered filtering of messages solves the EBI problems with scalability, while improving evolvability and reusability as well. Heavyweight connectors that include monitoring capabilities can be used to improve visibility and reduce the reliability problems of partial failure.

2. DISTRIBUTED OBJECTS

The distributed objects style organizes a system as a set of components interacting as peers. An object's state is completely hidden and protected from all other objects. This creates a well-defined interface for each object improving evolvability. Object systems are designed to isolate the data being processed. As a consequence, data streaming is not supported in general. However, this does provide better support for object mobility when combined with the mobile agent style.

3. BROKERED DISTRIBUTED OBJECTS

In order to reduce the impact of identity, modern distributed object systems typically use one or more intermediary styles to facilitate communication. This includes event-based integration and brokered client/server [11[12]. The brokered distributed object style introduces name resolver components whose purpose is to answer client object requests for general service names with the specific name of an object that will satisfy the request. Advantages are improved reusability and evolvability and the disadvantage is reduced efficiency.

4. REPRESENTATIONAL STATE TRANSFER (REST)

REST [1] describes the architectural style used to guide the development of the standard protocols that constitute the WWW architecture. REST, as a set of design choices, drew from a rich heritage of architectural principles and styles. There are six REST principles, or RPs: RP1: The key abstraction of information is a resource, named by an URL. Any information that can be named can be a resource presentation RP2: The representation of a resource is a sequence of bytes, plus representation metadata to describe those bytes. The particular form of the representation can be negotiated between REST components. RP3: All interactions are context-free-each interaction contains all of the information necessary to understand the request, independent of any requests that may have preceded it. RP4: Components perform only a small set of well-defined methods on a resource producing a representation to capture the current or intended state of that resource and transfer that representation between components. These methods are global to the specific architectural instantiation of REST; for instance, all resources exposed via HTTP are expected to support each operation identically. RP5: Idempotent operations and representation meta-data are encouraged in support of caching and representation reuse. RP6: The presence of intermediaries is promoted. Filtering or redirection

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

intermediaries may also use both the meta-data and the representations within request or responses to augment, restrict, or modify requests and responses in a manner that is transparent to both the user agent and the origin server.

IV. CONCLUSION

In this paper, we have discussed the characteristics of network based applications and presented a frame work for designing the architectures for network based applications. Also we have presented different architectural styles for network based applications. All the basic architectural styles and the derivative architectural styles from the basic one are discussed here. We compared all the architectural styles and discussed there advantages and disadvantages as well. REST provides a model not only for the development and evaluation of new features, but also for the identification and understanding of broken features.

Future work will focus on extending the architectural guidance toward the development of a protocol family using a more efficient tokenized syntax combined with the desirable properties identified by REST.

REFERENCES

[1] Roy Thomas. Fielding," Architectural Styles and the Design of Networked-based Software Architectures." Ph.D. dissertation, Information and Computer Science, University of California-Irvine, Irvine, CA. 2000.

[2] Dipanwita Thakur Banasthali and G.N. Purohit, "A Comparative Study on Software Architectural Styles for Network based Applications," IJCA, *Volume 47– No.20, June 2012.*

[3] D. E. Perry and A. L. Wolf.," Foundations for the study of software architecture", ACM SIGSOFT Software Engineering Notes, 17(4), Oct. 1992, pp. 40-52.

[4] A. S. Tanenbaum and R.van Renesse. "Distributed Operating Systems", ACM Computing Surveys, 17(4), Dec. 1985, pp. 419-470.

[5] D. Garlan and M. Shaw, " An introduction to software architecture", Ambriola & Tortola (eds.), Advances in Software Engineering & Knowledge Engineering, vol. II, World.

[6] G. Andrews," Paradigms for process interaction in distributed programs. ACM Computing Surveys, 23(1), Mar. 1991, pp. 49-90.

[7] A. Umar, "Object-Oriented Client/Server Internet Environments", Prentice Hall PTR, 1997.

[8] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network file system", In Proceedings of the Usenix Conference, June 1985, pp. 119-130.

[9] A. Fuggetta, G. P. Picco, and G. Vigna," Understanding code mobility", IEEE Transactions on Software Engineering, 24(5), May 1998, pp. 342-361.

[10] D. D. Clark and D. L. Tennenhouse," Architectural considerations for a new generation of protocols" In Proceedings of ACM SIGCOMM'90 Symposium, Philadelphia, PA, Sep. 1990, pp. 200–208.

[11] F. Buschmann and R. Meunier," A system of patterns", Coplien and Schmidt (eds.), Pattern *Languages of Program Design*, Addison-Wesley, 1995, pp. 325–343.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal." Pattern-oriented Software Architecture: A system of patterns" John Wiley & Sons Ltd., England, 1996.

[13] Amine Chigani **, "** Guiding Network-Centric Architectural Design: A Style based Approach", Dec 2007.

AUTHOR BIOGRAPHY

Mr. A. D. Devangavi is a graduate in M.Tech in Computer Science and Engineering from Visveswaraiah Technological University (VTU), Belgaum, Karnataka, India. He is currently working as Associate Professor in Information Science and Engineering Dept, Basaveshwar Engineering College, Bagalkot and has 18 years of teaching experience. Areas of research interest include Computer Networks, Operating Systems, Fuzzy Logic, OLAP and Software Architectures.