Volume 4, Issue 1, January 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Emerging Technologies: Network Architecture

Keshav Jindal*, Manoj Sharma, Dr. B.K. Sinha
Assistant Professor
NITRA Technical Campus
Ghaizabad, India

Abstract —This present work defines a framework for understanding software architecture via architectural styles and demonstrates how styles can be used to guide the architectural design of network-based application software. The first section defines introduction about network based applications are discussed. Second section is a framework for understanding software architecture through architectural styles including a consistent set of terminology for describing software architecture. The third section an architectural style for network-based applications is used to classify styles according to the architectural properties as they induce on architecture for distributed hypermedia.

Keywords: Hypermedia, Software based applications

1. INTRODUCTION

Network applications use a client-server architecture, where the client and server are two computers connected to the network. The client is typically a desktop, laptop or portable device like an Apple I-Phone. The server can be any of these, but is typically a computer in a data centre. In most (though not all) network applications, the client computer runs a Web client program like Firefox or Internet Explorer, and the server runs a Web server program like Apache or Internet Information Server. Shared data would be stored on the server or a computer it could access. Search engine companies like Google run client programs that constantly scan the Web, checking for new pages which can be indexed.

2. NETWORK BASED APPLICATIONS

Network Applications, Upgrades and Information Technology Once installed and running, we can use a computer network to share and back up files, play games, and run more sophisticated applications. Regular maintenance is essential to keep a network running well.

2.1 Types

P2P and File Sharing Networks

P2P (Peer to Peer) Internet file sharing systems use distributed network techniques and specialized protocols to support large-scale online file searching, sharing and messaging.

Virtual Private Networks (VPN) and Remote Access

A Virtual Private Network (VPN) supports local or long distance protected access to network resources by tunnelling through shared public links.

Internet Proxy Servers

Proxy servers are Internet gateways that provide network firewall, connection sharing, and caching technology for computers on a private network.

2.2 Constraints

Network Performance

The performance of an interconnection network in a massively parallel architecture is subject to physical constraints whose impact needs to be re-evaluated many times. Less Bandwidth and low capacity wires slows down the overall network performance. Two new performance-based design constraints are introduced: constant maximum throughput and constant unity queue. These new constraints are fundamentally different than previous constraints, which are based on some characterization of hardware implementation costs. Both of the new constraints allow performance and cost comparisons of different network configurations to be made on the basis of an equal ability to handle a range of traffic load.

Managing Network Storage

Storage networking utilizes specialized devices designed to serve or back up large amounts of data across local area networks. Two types of storage network technology exist - SAN (Storage Area Network) and NAS (Network Attached Storage). It has some disadvantages such as increased network workload, and inconvenience in disaster recovery. To overcome these disadvantages, we propose a channel-bonding technique and provide hot backup functions in the designed NAS system, named HUST server.

Internet Application Trends

Commonly used applications on the Internet today include Web browsers, email clients and instant messaging systems etc. Industry trends suggest that future network applications will continue evolving into less powerful environments without all these Internet applications.

3. ARCHITECTURES USED IN NETWORK-BASED APPLICATION

Architecture is found at multiple levels within software systems. This examines the highest level of abstraction in software architecture, where the interactions among components are capable of being realized in network communication.

Network-based vs. Distributed:

The primary distinction between network-based architectures and software architectures in general is that communication between components is restricted to message passing, or the equivalent of message passing if a more efficient mechanism can be selected at run-time based on the location of components. Another difference between distributed systems and network-based systems: a distributed system is one that looks to its users like an ordinary centralized system, but runs on multiple, independent CPUs. In contrast, network-based systems are those capable of operation across a network, but not necessarily in a fashion that is transparent to the user.

Application Software vs. Networking Software:

Application software architecture is an abstraction level of an overall system, in which the goals of a user action are represent able as functional architectural properties. For example, a hypermedia application must be concerned with the location of information pages, performing requests, and rendering data streams. This is in contrast to a networking abstraction, where the goal is to move bits from one location to another without regard to why those bits are being moved. It is only at the application level that we can evaluate design trade-offs based on the number of interactions per user action, the location of application state, the effective throughput of all data streams (as opposed to the potential throughput of a single data stream), the extent of communication being performed per user action.

Evaluating the Design of Application Architectures:

Architecture can be evaluated by its run-time characteristics, but we would obviously prefer an evaluation mechanism that could be applied to the candidate architectural designs before having to implement all of them. Unfortunately, architectural designs are notoriously hard to evaluate and compare in an objective manner. Like most anti facts of creative design, architectures are normally presented as a completed work, as if the design simply sprung fully formed from the architect's mind. In order to evaluate an architectural design, we need to examine the design rationale behind the constraints it places on a system, and compare the properties derived from those constraints to the target application's objectives.

Architectural Properties of Key Interest:

This section describes the architectural properties used to differentiate and classify architectural. It is not intended to be a comprehensive list. It included only those properties that are clearly influenced by the restricted set of styles surveyed.

Performance:

One of the main reasons to focus on styles for network based applications is because component interactions can be the dominant factor in determining user-perceived performance and network efficiency. Since the architectural style influences the nature of those interactions, selection of an appropriate architectural style can make the difference between success and failure in the deployment of a network-based application. The performance of a network-based application is bound first by the application requirements, then by the chosen interaction style, followed by the realized architecture, and finally by the implementation of each component.

Network Performance:

Network performance measures are used to describe some attributes of communication. Throughput is the rate at which information, including both application data and communication overhead, is transferred between components. Overhead can be separated into initial setup overhead and per-interaction overhead, a distinction which is useful for identifying connectors that can share setup overhead across multiple interactions (amortization). Bandwidth is a measure of the maximum available throughput over a given network link. Usable bandwidth refers to that portion of bandwidth which is actually available to the application.

User-perceived Performance:

User-perceived performance differs from network performance in that the performance of an action is measured in terms of its impact on the user in front of an application rather than the rate at which the network moves information. The primary measures for user perceived performance are latency and completion time.

Network Efficiency:

This essentially means that the most efficient architectural styles for a network-based application are those that can effectively minimize use of the network when it is possible to do so, through reuse of prior interactions (caching),

reduction of the frequency of network interactions in relation to user actions (replicated data and disconnected operation), or by removing the need for some interactions by moving the processing of data closer to the source of the data (mobile code).

Modifiability:
Modifiability is about the ease with which a change can be made to application architecture. Modifiability can be further broken down into resolvability, extensibility, customizability, configurability, and reusability, as described below. A particular concern of network-based systems is dynamic modifiability where the modification is made to a deployed application without stopping and restarting the entire system.

Portability:
Software is portable if it can run in different environments. Styles that induce portability include those that move code along with the data to be processed, such as the virtual machine and mobile agent styles, and those that constrain the data elements to a set of standardized formats.

Reliability:
Reliability, within the perspective of application architectures, can be viewed as the degree to which architecture is susceptible to failure at the system level in the presence of partial failures within components, connectors, or data. Styles can improve reliability by avoiding single points of failure, enabling redundancy, allowing monitoring, or reducing the scope of failure to a recoverable action.

4. NETWORK-BASED ARCHITECTURAL STYLES

This presents 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.

Classification Methodology:
The purpose of building software is not to create a specific topology of interactions or use a particular component type it is to create a system that meets or exceeds the application needs. The architectural styles chosen for a system's design must conform to those needs, not the other way around. Therefore, in order to provide useful design guidance, a classification of architectural styles should be based on the architectural properties induced by those styles.

Selection of Architectural Styles for Classification:
The set of architectural styles included in the classification is by no means comprehensive of all possible network-based application styles. Indeed, a new style can be formed merely by adding an architectural constraint to any one of the styles surveyed.

Style-induced Architectural Properties:
This classification uses relative changes in the architectural properties induced by each style as a means of illustrating the effect of each architectural style when applied to a system for distributed hypermedia. The architectural properties are relative in the sense that adding an architectural constraint may improve or reduce a given property, or simultaneously improve one aspect of the property and reduce some other aspect of the property. Likewise, improving one property may lead to the reduction of another.

Data-flow Styles:

Pipe and 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 processing them incrementally so that output begins before the input is completely consumed. This style is also referred to as a one-way data flow network. 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. 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. A filter cannot interact with its environment because it cannot know that any particular output stream shares a controller with any particular input stream.

Uniform Pipe and Filter (UPF):
The uniform pipe and filter style adds the constraint that all filters must have the same interface. The primary example of this style is found in the Unix operating system, where filter processes have an interface consisting of one input data stream of characters (stdin) and two output data streams of characters (stdout and stderr). 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.

Replication Styles:

Replicated Repository (RR):
Systems based on the replicated repository style 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. Distributed file systems, such as XMS, and remote versioning systems, like CVS [www.cyclic.com], are the primary examples. 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.

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, as in the WWW, or where complete access to the repository is unnecessary. Lazy replication occurs when data is replicated upon a not-yet-cached response for a request, relying on locality of reference and commonality of interest to propagate useful items into the cache for later reuse. Active replication can be performed by prefetching cacheable entries based on anticipated requests.

Hierarchical Styles:
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. A client is a triggering process; a server is a reactive process. Clients make requests that trigger reactions from servers. Thus, a client initiates activity at times of its choosing; it often then delays until its request has been serviced. On the other hand, a server waits for requests to be made and then reacts to them. A server is usually a non-terminating process and often provides service to more than one client.

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. Although layered system is considered a "pure" style, its use within network-based systems is limited to its combination with the client server style to provide layered-client-server. Layered systems reduce coupling across multiple layers by hiding the inner layers from all except the adjacent outer layer, thus improving resolvability and reusability. Examples include the processing of layered communication protocols, such as the TCP/IP and OSI protocol stacks, and hardware interface libraries. The primary disadvantage of layered systems is that they add overhead and latency to the processing of data, reducing user perceived performance.

Mobile Code Styles:
Mobile code styles use mobility in order to dynamically change the distance between the processing and source of data or destination of results. These styles are comprehensively examined in Fuggetta. A site abstraction is introduced at the architectural level, as part of the active configuration, in order to take into account the location of the different components. Introducing the concept of location makes it possible to model the cost of an interaction between components at the design level. In particular, an interaction between components that share the same location is considered to have negligible cost when compared to an interaction involving communication through the network. By changing its location, a component may improve the proximity and quality of its interaction, reducing interaction costs and thereby improving efficiency and user-perceived performance.

Virtual Machine (VM):
Underlying all of the mobile code styles is the notion of a virtual machine, or interpreter, style. 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. It is not, in itself, a network-based style, but it is commonly used as such when combined with a component in the client-server style (REV and COD styles). Virtual machines are commonly used as the engine for scripting languages, including general purpose languages like Perl and task-specific languages like PostScript. The primary benefits are the separation between instruction and implementation on a particular platform (portability) and ease of extensibility. Visibility is reduced because it is hard to know what an executable will do simply by looking at the code. Simplicity is reduced due to the need to manage the evaluation environment, but that may be compensated in some cases as a result of simplifying the static functionality.

Remote Evaluation (REV):
In the remote evaluation style, 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 (CPU cycles, data source, etc.) required, which happen to be located at a remote site. 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 remote evaluation style assumes that the provided code will be executed in a sheltered environment, such that it won't impact other clients of the same server aside from the resources being used.
Mobile Agent (MA):
In the mobile agent style, 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 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. An application can be in the midst of processing information at one location when it decides to move to another location, presumably in order to reduce the distance between it and the next set of data it wishes to process. In addition, the reliability problem of partial failure is reduced because the application state is in one location at a time.

Peer-to-Peer Styles:

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. The event-based integration style provides strong support for extensibility through the ease of adding new components that listen for events, for reuse by encouraging a general event interface and integration mechanism, and for evolution by allowing components to be replaced without affecting the interfaces of other components. The basic form of EBI system consists of one event bus to which all components listen for events of interest to them.

Distributed Objects:

The distributed objects style organizes a system as a set of components interacting as peers. An object is an entity that encapsulates some private state information or data, a set of associated operations or procedures that manipulate the data, and possibly a thread of control, so that collectively they can be considered a single unit. In general, an object's state is completely hidden and protected from all other objects. The only way it can be examined or modified is by making a request or invocation on one of the object's publicly accessible operations. This creates a well-defined interface for each object, enabling the specification of an object's operations to be made public while at the same time keeping the implementation of its operations and the representation of its state information private, thus improving resolvability.

5. CONCLUSION

This examined the scope of the focusing on network based application and architectures and describing how styles can be used to guide their architectural design. It also defined the set of architectural properties that will be used throughout the comparison and evaluation of architectural styles. It presents 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.

REFERENCES

[1] M. Jackson. Problems, methods, and specialization. *IEEE Software*, 11(6), [condensed from *Software Engineering Journal*], Nov. 1994. pp. 57-62.
[2] J. Q. Author, "Title of the paper," *Journal*, vol. 10, pp. 1-20, June 1993.
[3] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979.
[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Mass., 1998.
[5] D. Flanagan. *JavaScript: The Definitive Guide, 3rd edition*. O'Reilly & Associates, Sebastopol, CA, 1998.
[6] D. Flanagan. *JavaTM in a Nutshell, 3rd edition*. O'Reilly & Associates, Sebastopol, CA, 1999.
[7] H. Zimmerman. OSI reference model -- The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28, Apr. 1980.
[8] A. Umar. *Object-Oriented Client/Server Internet Environments*. Prentice Hall PTR, 1997.

