Module: SE Seminar 7

Seminar 7 Distributed Software Engineering and Service-Oriented Architecture

Study Chapter 18 and 19

Seminar 7 provides you with an understanding of the fundamentals of designing and implementing distributed software systems. The idea of software as a service will be introduced. Web services and service-oriented application development are studied.

Part 1: Distributed System Architectures

Business e-environments are evolving such that it is often impossible to know who is doing what and where. Today, many of our corporations operate globally, 24 hours a day, seven days a week. We see a rapid melding of a global economy. Work and tasks are quickly updated and changed based on global information, often reducing business cycle times to a fraction of previous work-time. The term, information economy, has been used to describe the integration of communication and data technology into information technology using the Internet as the communication method.

Global corporations can only work through information technology, and the only way technology can work is through information systems which are readily available, flexible to changing environments, regularly monitored, and represent the "real world". Distributed systems technology allows corporations to create the information systems which can be adapted for the needs of 21st century business.

It would be difficult to find any large computer-based system today without some type of distributed system. Most information systems distribute processing over several processors. Worldwide, the top priority in the next decade will be updating architectures for implementing or improving distributed computing platforms.

The primary characteristics of a distributed system is the ability to share resources, run programs concurrently, and provide systems to run on large and small systems. They also must have a high level of fault tolerance and the ability to hide the distributed processing from the user (transparency).

In 1994, Peter Deutch published 8 fallacies of Distributed Computing. For years Distributed Data Services were built with these in mind:

1. The network is reliable

2. Latency is zero
3. Bandwidth is infinite
4. Network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

We have learned a lot in 15 years. We now know that we need to revisit Deutch's list. We are savvier on what it takes to implement a distributed system. The evolution of distributed software engineering has had a significant impact on the implementation of distributed systems. Distributed systems include files systems, database systems, operating systems, and the Internet.

The characteristics of the DDS include the hardware, operating systems, languages and applications, middleware (abstract level between software components), and the virtual machines which communicate with each other.

Distributed Systems offer different configurations for building their concurrency – simultaneous users on many computers. All, however, need to be scalable where resources can be added or deleted. Another key factor for DDS is the ability to provide a high level of security. This includes securing data from unauthorized entry, denial of attacks, data attacks which affects data integrity. There also needs to be a high level of fault tolerance – recovery from failures (using hardware and data redundancy and software rollbacks).

The following sections look at different characteristics of computer systems and how they are addressed in distributed systems.

Concurrency

Concurrency is defined as multiple jobs being run at the same time. The key issue in a DDS is to ensure the retrievals and updates are done on current instances of the data. Otherwise, it results in inconsistencies and poorly informed decisions. In a distributed system, there needs to be processes in place so users do not affect the work of other users. Another factor in concurrency is process synchronization of resources. The traditional method and one which continues to work if implemented correctly is the use of hardware or software locks. In a distributed system, managing transactions is another way to manage concurrency. (Anderson, 2008)

Transactions

Transactions were discussed in an earlier seminar. Here, we will look at it from the distributed system point of view. A transaction is a unit of work. It completely finishes or it is not done at all. This guarantees the integrity of the data.

A transaction includes several instructions to complete a task in a multi-user environment. It has some of the same characteristics as a transaction in a database, but when we move it to a distributed environment, we add external networking communication problems and processing problems. In a distributed system, transactions are used for managing all resources. A 2-phase lock mechanism is used to guarantee serialbility. Each data item has a lock. Before the transaction can access the item, the system checks for a lock. If it is attached, the system locks out requests for the data until the lock is removed. There are two types of locks – read and write locks. A 2-phase mechanism allows read locks to be released anytime the process is done and forces the release of write locks to wait until the transaction is done.

Once a transaction begins, it ends either in a Commit (resources are released, data is saved, and the system returns to a consistent state) or Abort (all resources return to previous state).

Transparency

Transparency is important in distributed systems because it hides the complexities from the user. They do not want to know where the data is stored, only that their query returns the correct data. They don't want to know how others in the system are changing the data, only that the data they are looking at is current and correct.

Goel (2004) detailed the types of transparency needed for distributed systems.

a. Access transparency – hides the management of files stored within the system.
b. Location transparency – hides the physical location of information in the system.
c. Concurrency transparency – hides the mechanisms that guarantee concurrency.
d. Replication transparency – hides the process of replicating data and how the data is stored.
e. Failure transparency – allows the system to recover without the users being aware of the failure.
f. Mobility transparency - allows mobile users to resources without knowing where the resources are.
g. Performance transparency – completes automatic load balancing without clients being aware.
h. Scaling transparency – hides the process of growth to the applications.

As can be seen in our list, the key to transparency is, from the user's point of view, all the technical details are hidden under a layer of abstraction.

Fault Tolerance

Faults and Failures were discussed in an earlier seminar. The distributed system is more resistant to failure and recovers more quickly because there are multiple sites that provide the redundancy to a faster recovery.
A extreme example can be seen looking at the companies who had their headquarters in the twin towers. Those with redundant systems in other geographic locations recovered quickly. This was also seen in New Orleans after the Katrina hurricane. Those fully distributed lost resources but the data was recovered quickly.

Distributed Systems Security

There are different approaches to handling security in a distributed system. The SANS Institute assessed three different environments which provide good examples for how security is addressed in distributed systems – Java, CORBA, and COM+.

Java

Java's architecture has unique characteristics to build distributed systems having a high degree of security. Java programmers can use the Java language to build systems to remotely execute on distributed systems. Java's architecture builds these more secure systems because the source code is translated into byte code which, in turn, is interpreted by the Java Virtual Machine(VM). Inherent in the systems are security checks between the client and the remote server. Security checks are implemented between the remote server and the client executing the application. One approach is a "sandbox" security model implementing security using Java's APIs (Figure 1). The VM creates this secure "sandbox" where the programs execute. The sandbox performs a variety of checks at runtime inclreasing the security of the system. (Moreno, 2002)

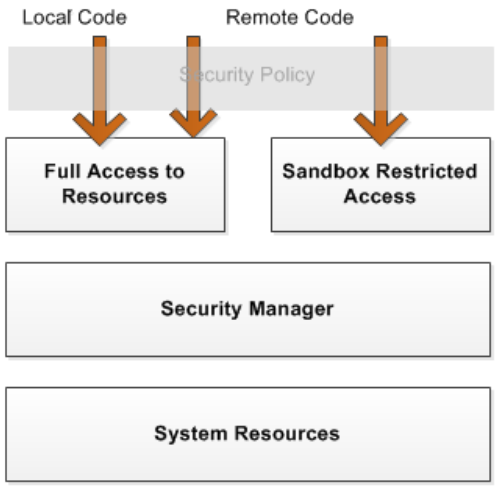

Figure 1. The Java Security Model (based on Moreno, 2002).

CORBA

CORBA (Common Object Request Broker Architecture) was created by the Object Management Group (OMG). CORBA allows objects from different languages and users to communicate. To do this, CORBA has an Interface Definition Language (IDL) creating the bridge between all the interacting objects and users. Information is sent to the IDL mapping the requests to the appropriate user. "Each object instance has a unique entry in the Object Request Broker (ORB) which handles requests between objects and between user and objects. Objects with similar security requirements are grouped into domains, and a security policy is applied to the domain, which is enforced by the ORB. Communication between ORBs is handled by bridges, gateways, and inter-ORB protocols (Figure 2)." (Moreno, 2002).

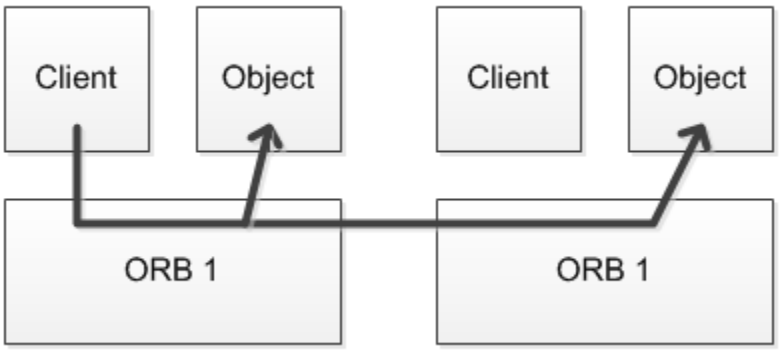

Figure 2 – CORBA Model (Moreno, 2002)

COM+

COM+ is Microsoft's distributed architecture. "Clients run an application that connects to a server with running COM+ services. This server, in turn, connects to the back-end servers." (Moreno, 2002). With this architecture clients are able to share resources on the COM+ server. They can use different languages. Since the COM+ servers have the implemented security, the developers don't need to worry about building security into their applications.

Client-Server Architectures

In the client-server architecture, as with client/server design, application processing is divided into *services* that are provided by servers and *clients* using these services. The clients know of servers but servers do not need to know about the clients. Clients and servers are the logical processes in the system.

The two implementations of a two-tier client-server model are the thin-client model and the fat-client model.

Thin Client

In a thin-client model, all the processing and managing of the data are completed by the server. The client is responsible only for running the software. This model allows us to store data centrally while efficiently distributing information and applications to clients. A major advantage to the thin client is the ability to have inexpensive computing boxes (some lower than $100 running Linux) on the client side.

Fat-Client

In this model, the server is only responsible for maintaining and managing the data. The software residing on the client machine executes the applications and manages the interactions with the system user.

New platforms have provided a way to build a hybrid architecture where pieces of the application (such as Java applets) can be downloaded to the client while most of the processing continues to be on the server.

The two types of architectures are the two-tier and three-tier architecture. In the two-tier, the user interface is on the client and the database management services are on the server.

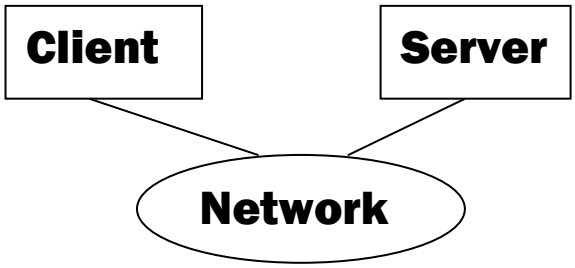

Figure 3. two-tiered client-server architecture.

In the three-tier architecture, the middle tier is between the client and the server and has different functions based on the needs of the system. It may be used for executing applications, queuing, and/or database applications. The three-tier architecture reduces the flexibility of a system but does improve the productivity. Thus, the choice is based on the individual needs. (Sadoski, 1997)

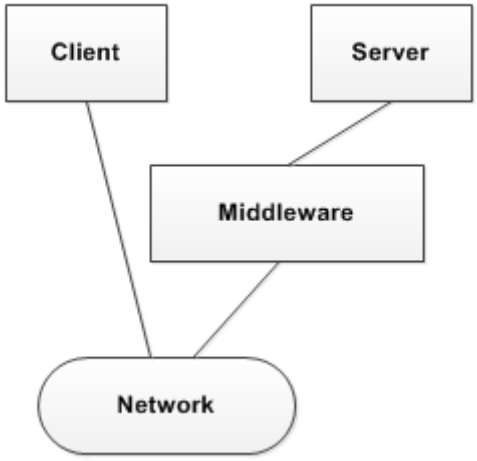

Figure 4. three-tiered client-server architecture.

Summary - Advantages of Client/Server Environment

The client/server environment allows organizations to maximize the benefits of micro-technology. PCs can deliver extraordinary computing power at a fraction of cost of mainframes.

The architecture allows most of the processing to be done close to the data source and, thus, facilitates efficient processing. It also facilitates the use of intelligent user interfaces. (Oracle, 2003)

Distributed Architecture Patterns

Designing distributed systems is a complex process. Not only do we have heterogeneous hardware systems, but we also have diverse software and human systems. It is a difficult task. Over the years, several strong architectural patterns have emerged which support the implementation of distributed systems.

Master/Slave Architecture

This pattern has two objects: one Master and many Slaves. The master sends the work (set of computational processes) to a shared structure (such as a queue) where the "slaves" pick up the jobs and process them. This pattern does a good job of load balancing in a distributed system because the workers distribute the load and is discussed in the text.

Distributed Component Architecture

Distributed object-oriented systems have an architecture that blends object-oriented and client/server technologies. Distributed objects blend the distribution advantages of client/server technology with the real-world information contained in object-oriented models. (Sutherland, 1997)

Distributed component architecture allows objects to be distributed across diverse networks and allows each part to be integrated into a single system. Modeling business processes is not a matter of modeling organizational charts and company policy manuals. Real system modeling requires a model representing the way work is actually accomplished, and the ways business is really conducted. This architecture models the real business entities and operations to accomplish work and produce value. Object orientation was developed in the 1960s to provide the capability to build models reflecting real systems. (Cetus Team, 2004) CORBA and Java distributed systems can be configured as distributed components.

A relatively new concept having an impact on recent distributed object architectures is the use of Software Agents. A software agent is a computer program (object) acting "intelligently" to make decisions based on previously defined preferences. The uniqueness of the agent is in the way it communicates with other objects – using high-level declarative messages. (Purvis, Cranefield, and Ward, 1998) Some researchers have made the case that Corba objects are actually agents because they have the ability to be "reactive, autonomous, social, and pro-active" -- also the definition of a software agent. Research into the area of Software Agents is just beginning

Peer-to-Peer Systems

We cannot complete a lecture on distributed systems without including something about Peer-to-Peer (P2P) systems. P2P systems are those distributed systems which allow servers to communicate with each other through a network. Many of them are set up in a decentralized manner such as Gnutella and Freenet while other are more centralized. Most can be categorized with a hybrid approach because part of the functionality is still on centralized servers somewhere. For example, Napster uses special servers that perform unique functions, such as a server to hold indexes of files (Yang and Garcia-Molina, 2001).

Service-Oriented Architecture

Most organizations are facing a crisis in integrating their technology to work seamlessly together to provide value-added services. Service-oriented architectures were designed to help solve some of these problems. In service-oriented architectures, the key is to provide high-level services to integrate disparate technologies. One such approach is the use of web services. Web Services use something like XML to create connections

between service providers and service consumers. The three main goals of SOA are to build for reuse, build for agility, and build to align with business/IT goals.

More discussion on Service-Oriented Architecture is found in Part 2 of the lecture. The need for this architecture will continue to grow as organizations become more complex and geographically dispersed.

Part 2. Service- Oriented Architecture

Service-Oriented Architecture (SOA) was introduced in the previous section. As networks became more stable and the speed became acceptable for production work, we saw a new trend in distributed computing – web services. This is discussed in detail in the WA Module. The key is the modularization of software application functions that can be created as services for other applications. This has been done with a high level of abstraction where applications can request a service without knowing how the service is implemented; only what its basic task is.

Hashimi (2003) has done significant research on SOA. He defined three key characteristics of a service:

1. Platform-independence
2. Dynamically located (implementing location transparency)
3. Service-independence – "maintains its own state".

The example of a service in Figure 5 shows how these work;

Suppose I want an on-line tax service to aid in completing my taxes. I query a directory service to find a list of service providers to do taxes online. Based on a set of criteria, I select a service (Figure 5).

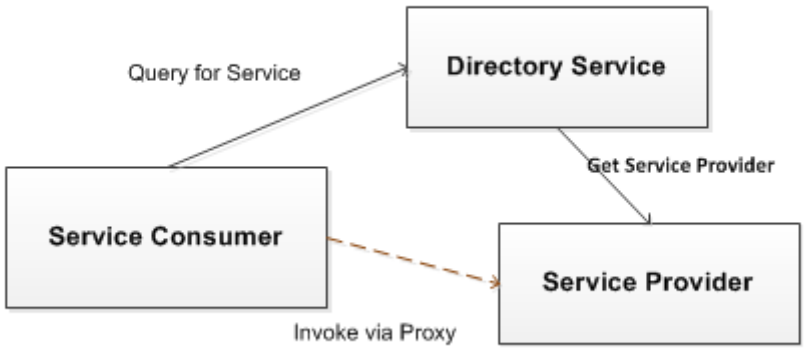

Figure 5 Directory service (based on Hashimi (2003)

While web services are concerned with application specifications, SOA is the design approach for using the services. It allows us to create an architecture that models the connections between the web services and location transparency.

SOAs are implemented in a variety of environments: Web Pages, Java IDEs, .NET. All have common characteristics:

1. They have independent interfaces using XML.
2. They have a web services language to explain the tasks of the services.
3. The communication is done with an XML Schema
4. The services are managed by a registry (a directory list) in the organization
5. Each service as certain requirements attached called quality of service (QoS), primarily for security purposes.

The text provides a good overview for how services are implemented. The next sections in the lecture address SOA trends.

SOA and Reusable Components

An ongoing theme of the course has been to leverage what we already know to improve the productivity and quality of the applications that we build. One of the main ways of doing this is to utilize reusable components, as we mentioned in earlier weeks. SOAs can effectively supply these reusable components. Since the services in any SOA are, by definition, components, and service providers are working toward building as many services as they can, the contribution to reuse can be enormous. In order for this to be realized, services must be independent, easy to select based on developer needs, and easy to integrate into an enterprise solution.

One way to manage services for reuse is to create a *centralized service center*. Using this architecture, we organize services in a database which can be searched based on criteria. The service specification is a critical piece since this determines whether the service fits the needs. Services are then provided to clients based on their criteria. Because of the complex relationships between services and their integration into existing systems, such a service center needs significant computing power to manipulate large amounts of data and to provide the integrating services.

Another approach could be a *distributed service center*. Here, we still have the ability to provide a database for services, but the computing is distributed. Thus, we can have many services being provided concurrently. The important factor for success is high bandwidth. From the developer's point of view, it still appears as a service center.

No matter how we set up the infrastructure, services are currently a major trend in software reuse.

Service Engineering

There are many independent factors to take into account when providing a successful 'service". Thus, it takes an engineering approach to attempt to build a solid solution. As with other activities we have discussed for designing of software applications, design patterns can have a positive impact on the effort required to "engineer" a SOA.

There are a several good sources for finding design patterns. However, one of the best sources is OASIS (2010). OASIS is a non-profit consortium for the advancement and standards for Open Source. One of their committees, SOA Adoption Blueprints, has created sample blueprints you can download which includes best practices and patterns for SOA architects.

Again, your text has good examples for implementing a service. There are, however many independent factors which can either help in its success or lead to failure.

The external factors to be considered in any SOA are:

1. Where the service will be hosted and what types of services are provided – data services, integration services, etc
2. What infrastructure is in place for services to call other services
3. Network characteristics that affect the ability to provide the services – load balancing, etc
4. What security procedures are in place
5. What transactional services are provided
6. What day-to-day management activities are provided (Vektrel, 2008)

These factors build the infrastructure to house the services.

Although we have been working with SOAs for several years, it has only become fully realized as a major solution in the last 5 years. The advantages that web services have brought to the IT industry are significant. "Service-driven enterprises can achieve increased efficiency via greater reuse of IT assets, faster delivery of value to the business and improved adaptability to gracefully support any expected and non-expected changes" (Zhu, 2005).

Modeling SOAs

Modeling SOAs starts by modeling three different entities; the service provider, the service consumer, and the service broker. We saw a glimpse of what this would look like in Figure 5.

An enterprise modeling approach includes all the parts of an organization that make up the enterprise solution. An example of an enterprise model is shown in Figure 6. Each of the services could be done in-house or outsourced.

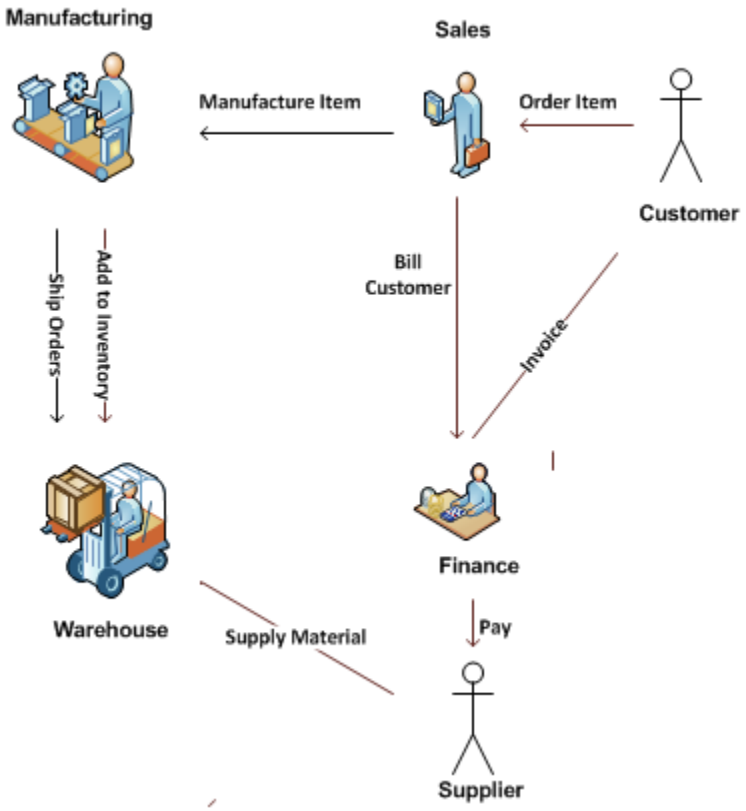

Figure 6. Enterprise Model

To fully model the entire operation, the services represented in figure 6 (Sales, Manufacturing, Warehouse, and Finance) would need to be decomposed into greater detail. Jones and Maynard (2005) provide a very good example of how we can create the complete model.

Summary

The driving force behind SOA is the need to provide software developers with the ability to manage and address the ever-changing technology and business needs. SOAs give us the ability to create applications consisting of independent software components tied together with web services. Thus, the greatest benefit we get from SOA is the flexibility to respond to changes. The statistics resulting in the implementation of SOA are very good – they demonstrate significant cost-savings and customer satisfaction.

References and Additional Reading

Anderson, R. (2008) Chapter 6 Distributed Systems. **Security Engineering: A Guide to Building Dependable Distributed Systems.** Wiley & Sons. 2nd Edition. ISBN 10:0470068526 http://www.cl.cam.ac.uk/~rja14/Papers/SE-06.pdf

Barry, D. (n.d.). Web Services and Service-Oriented Architectures. http://www.service-architecture.com/index.html

Cetus Team.(2002) Distributed Objects & Components: Business Objects. Nov **http://www.objenv.com/cetus/oo_business_objects.html**

Chung, K., Zimmerman, P. Chandra , S. (2006) What is a thin client? Searchnetworking.com. March. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci213135,00.html

CORBA (2009.) Homepage**. http://www.corba.org/** [Accessed August, 2010]

Goel, A (2004) Advances in Distributed Systems An Introduction. http://www.eecg.toronto.edu/~ashvin/courses/ece1746/2004/introduction.pdf

Hashimi, S (2003). Service-Oriented Arcitecture Explained. O'REILLY. http://ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html

Jones, S & Maynard, P (2005) A Methodology for Service Architectures. London © Capgemini 2005 ***http://www.oasis-open.org/committees/download.php/15071/A%20methodology%20for%20Service%20Architectures%201%202%204%20-%20OASIS%20Contribution.pdf***

Moreno, A (2002). Distributed Systems Security: Java, CORBA, and COM+. SANS Institute. http://www.sans.org/reading_room/whitepapers/application/distributed-systems-security-java-corba-com-plus_28

OASIS (2010). OASIS Service Oriented Architectural Adoption Blueprints TC. Organization for the Advancement of Structured Information Standards**. http://www.oasis-open.org/committees/soa-blueprints/faq.php**

Oracle. ,(2003) Application Architecture: Introduction to Client Server Environment. http://www.stanford.edu/dept/itss/docs/oracle/10g/server.101/b10743/dist_pro.htm

Purvis, M. , Cranefield, S., and Ward, S. (1998) Distributed Software Systems: From Objects to Agents. 1998 IEEE. Published in the Proceedings of SE:EP'98, January 1998 Dunedin, New Zealand. Digital Object Identifier: 10.1109/SEEP.1998.707646

Sadoski, D. (1997) Client/Server Software Architectures--An Overview SEI http://www-rcf.usc.edu/~anthonyb/itp320/clntsvrov.doc

Vektrel LLC (2008). SOA Service Engineering. http://www.vektrel.com/Topics/SoaServiceEng/SoaServiceEng.html

Yang, B & Garcia-Molina (2001). Comparing Hybrid Peer-To-Peer Systems. Proceedings of the 27th VLDB Conference, Roma, Italy, 2001. http://www.dia.uniroma3.it/~vldbproc/060_561.pdf

Zhu, H. (2005) . Building Reusable Components with Service-Oriented Architectures Paper presented at the 2005 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2005), Las Vegas, Nevada, USA

Reading Assignments

The assigned readings for this seminar are Chapter 18, Chapter 19.