



Designing Highly Available Web-Based Software Systems

Michael Acton
Lockheed Martin Mission Systems

Highly available Web-based applications require designs that address issues not only of availability, but also of reachability and performance. This article defines high availability and presents approaches for producing maintainable, highly available Web applications. This author cites the Global Combat Support System-Air Force, a network-centric, common technical services-based highly available system as one such example.

To guarantee the quality of service (QoS) deployed warfighters need, combat support systems and applications should be designed to provide near 100 percent availability. Each design activity, which includes usability design, functional design, role-based access control design, and component design, supports the warfighter in some meaningful way. When dealing with the vast technical challenges related to combat support Web application design, it is important not to lose sight of this overarching objective.

The Global Combat Support System-Air Force (GCSS-AF) Air Force Portal, shown in Figure 1, is an example of a Web application that provides dynamic personalized content to warfighters. The Air Force Portal provides both functional and informational capabilities; warfighters can perform their mission using the hosted combat support mission applications, access U.S. Air Force resources and tools, and read about current events. Critical combat support mission applications

hosted on the GCSS-AF system, like the Online Vehicle Interactive Management System, Fleet Asset Status, and the Combat Ammunition System, provide excellent examples of the types of Web applications that should be designed for high availability.

One aspect of application development that is often overlooked is ensuring that applications are designed to provide high availability (HA). Proper attention must be given to HA during application design and integration testing to avoid fielding applications that cannot provide 100 percent up time. To illustrate this point, consider the testing phase, which occurs late in development. Why is it that, when compared to functional testing, which usually receives proper attention, HA testing is scarcely considered?

There are many reasons why HA does not receive proper attention during the design phase. One reason is that engineering teams are rushed to meet schedules and do not have the time for proper HA

design. Another reason is that many software engineers do not have experience with HA construction and instead focus primarily on functionality. Finally, Web application development is a relatively new paradigm; at least the aspect of developing for a shared enterprise is new.

The remainder of this article defines HA, presents concepts of HA system design, and introduces design approaches for producing maintainable, highly available Web applications.

HA Described

As viewed, HA is comprised of three key components: *availability*, *reachability*, and *performance*. If any of these components is deficient or fails, then the Web-based system is essentially rendered degraded, or worse, unusable to the warfighters depending on its services.

A system is *available* when all of the necessary system services are up and operating correctly. When the system is available, users can log in through the security subsystem, navigate the Web site, and access and use any of the hosted Web applications for which they have permission. Required services are those that must be operating for a user to receive service. If just one *required* service goes down, the entire system moves into a state of non-availability and becomes incapable of providing *any* service to users. For example, every essential service of the system may be alive except the security subsystem user-authentication service. Without this one crucial service, the system is rendered non-operational. However, system availability is only one element of HA, the system must also be both reachable and performing adequately.

A system is *reachable* when users can access it over the network using a Web browser. Reachability is primarily concerned with assuring robust network service. Problems such as network service outages, improperly configured firewalls and proxies, and router failures will make even an available system unreachable [1].

Figure 1: The GCSS-AF Air Force Portal at <https://www.my.af.mil>



If the system is both available and reachable, users will receive service, and hopefully with good performance.

For a Web application to be in a highly available state, availability and reachability must be provided in a redundant manner that effectively removes all potential *single points of failure*. Fail-over is a system capability that allows a failed service to automatically be recovered (usually on a secondary or redundant server) in such a manner that the impact on both system processing and users' work is negligible. To ensure fail-over, every server, software service, and network component comprising the system must have at least two independent instances configured. In this manner, users cannot be denied service by any single system component failure.

The *performance* of HA Web applications must, at a minimum, adequately support warfighters in accomplishing their mission. Poor performance hinders user efficiency, creates frustration, and ultimately degrades U.S. Air Force operational capability. Anybody that has ever used a dial-up modem to connect to the Internet has experienced poor performance due to phone line throughput limitations (see Figure 2). T1 network lines provide markedly faster data transfer rates. Imagine warfighters deployed in the desert trying to use a dial-up connection to access critical combat support capabilities.

A monitoring capability for HA systems and applications must be designed as part of the overall solution. Using the system's HA requirements, a comprehensive set of metrics must be established, monitored, and reported to ensure that all aspects of the systems are meeting the required HA thresholds. HA testing is best accomplished by using proactive monitoring tools that actually perform the same actions performed by users. Simple ping-style monitoring tools are not sufficient because they are only capable of checking the system health at the operating system level. These are not capable of monitoring any application-level services. Some important metrics to consider tracking are: overall aggregate system availability (which considers all system services necessary to provide service); overall aggregate system reachability from several locations (i.e., several different bases); and average round-trip time for heavily used capabilities (i.e., a commonly used transaction performed via a Web application).

HA System Design

In order to design HA applications, developers must understand the HA

capabilities that the hosting system allows. Two of the most critical design aspects of HA systems are the physical system architecture and session clustering. In order to design HA Web applications, it is necessary to understand how the hosting system implemented these design characteristics.

GCSS-AF has been designed as a highly available system, and affords services that allow Web applications to be hosted on the system in an HA configuration. Availability, reachability, and performance have been painstakingly designed and implemented in the GCSS-AF system to ensure that it performs adequately and that no single point of failure exists. This means that every aspect of the system is immune from any single system component failure.

Every physical server and software service of GCSS-AF is carefully examined during the architecture phase to ensure that at least two independent *silos* are built and configured in such a way that if any one silo were to go down, the remaining silo(s) would pick up the workload. The *intent* is to provide users a system where they do not experience any interruption of service.

Figure 3 illustrates a simplified, tiered, physical-system architecture suitable for hosting HA Web applications (the GCSS-AF architecture is significantly more complex, both in terms of tiers and services). Three Web servers and three Web application servers are interconnected to form a

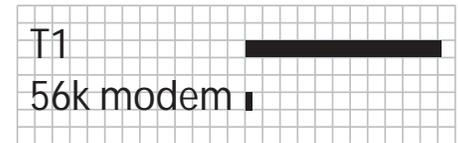


Figure 2: T1 Network Connections are Capable of Rates of 1.544Mbps Versus a Dial-Up Modem That Is Limited to 56Kbps

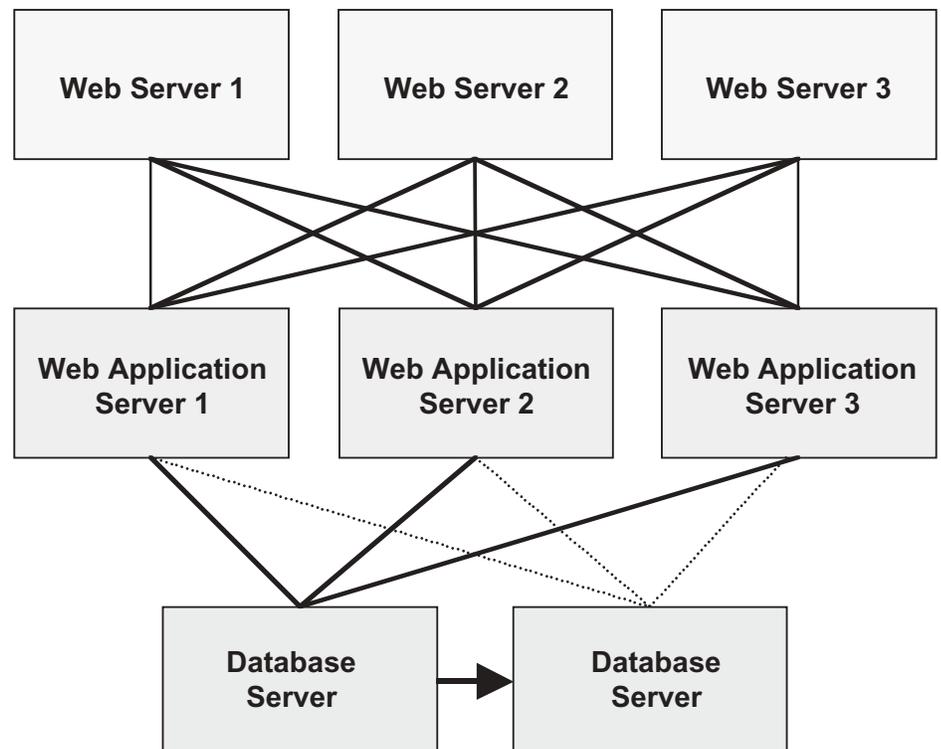
set of redundant services. In the event of any system component failure (i.e., server crash), these networked servers automatically reroute user requests to an available Web application server.

Figure 3 also illustrates simple *fail-over* at the database server tier. Should the primary database server fail, the *hot standby* database takes over and provides service to maintain availability. In this case, the database servers have been configured in a server cluster. The clustering software runs on each of the database servers and can determine when a failure occurs. When this clustering software determines a failure, it promotes the hot standby server to the master database server.

In addition to fail-over, this configuration also provides load balancing of user sessions (at the Web and Web application server tiers). Load balancing coupled with fail-over provides a better operational solution than fail-over alone because it provides for greater scalability and takes advantage of all available capacity, whereas a hot standby idles until needed.

In fact, all GCSS-AF system components have been designed using the HA

Figure 3: Simplified Tiered System Architecture for Hosting HA Web Application



principle of eliminating every potential single point of failure. At the network layer, reachability and network performance are provided by modern redundant routers and switches, which ensure multiple access paths to the system. The system is made up of state-of-the-art redundant hardware components that provide exceptional performance and availability (GCSS-AF employs SunFire technology).

At the application layer, performance and availability are provided by redundant software services, which are hosted on different servers to prevent a server failure from impacting software service availability. Additionally, each software component has been designed and configured to support HA optimally. None of this happened by chance; this was all considered in the earliest phases of design.

The second crucial topic relative to HA system design is user session fail-over assurance. A user session is nothing more than a set of related requests and responses between the user and a Web application server. A session is established when a user first accesses the Web application, and is maintained until the user logs off. If for any reason the user session is lost, the Web application server will no longer be able to associate the user with the work they were doing; all of the user's unsaved work will be lost.

The most common causes of user session loss are server and required software service failures (usually in the form of crashes or corruptions). Although the architecture depicted in Figure 3 does provide multiple paths for traversing between tiers, it does not automatically provide user session fail-over. Session fail-over

must be designed into the system.

There are essentially three session management approaches used in Web application servers: no session fail-over, database persistent sessions, and memory-to-memory sessions [2]. No session fail-over is unfortunately the most common, and is the default for most Web application servers. In the event of a server failure, all user sessions will be lost along with any unsaved work. These users will be forced to log on again and start over from scratch.

The second approach, database persistent sessions, requires session data to be stored and maintained in a database while the session is alive. If the server fails, then

“Why is it that, when compared to functional testing, which usually receives proper attention, HA testing is scarcely considered?”

the user's session will be transferred transparently onto another Web application server. However, a performance drawback exists with the database approach – session data must be saved and retrieved from the database in order to maintain the session state.

The third approach, memory-to-memory sessions, is the best approach. In this approach, session data is replicated among

the servers, providing session fail-over and much better performance than the database approach. Because sessions can be transferred among any of the servers in the cluster, both the database and the memory-to-memory session techniques are referred to as *session clustering* approaches. *Session clustering* is required for HA systems like GCSS-AF.

Although session clustering is documented as an open standard in the Java Servlet 2.3 Application Program Interface (API) Specification, clustering does not generally provide fail-over across application servers in different vendor implementations. For example, a Web application hosted on IBM WebSphere cannot fail-over to the Oracle 9iAS and vice-versa. Therefore, session clustering must be addressed separately for each supported application server type (GCSS-AF supports four different Web application servers: IBM WebSphere, Microsoft IIS, Oracle 9iAS, and BroadVision IM; each is independently configured for HA).

Now that some of the important aspects of HA system design have been addressed, we delve into the design considerations for HA Web applications.

Web Application HA Design Considerations

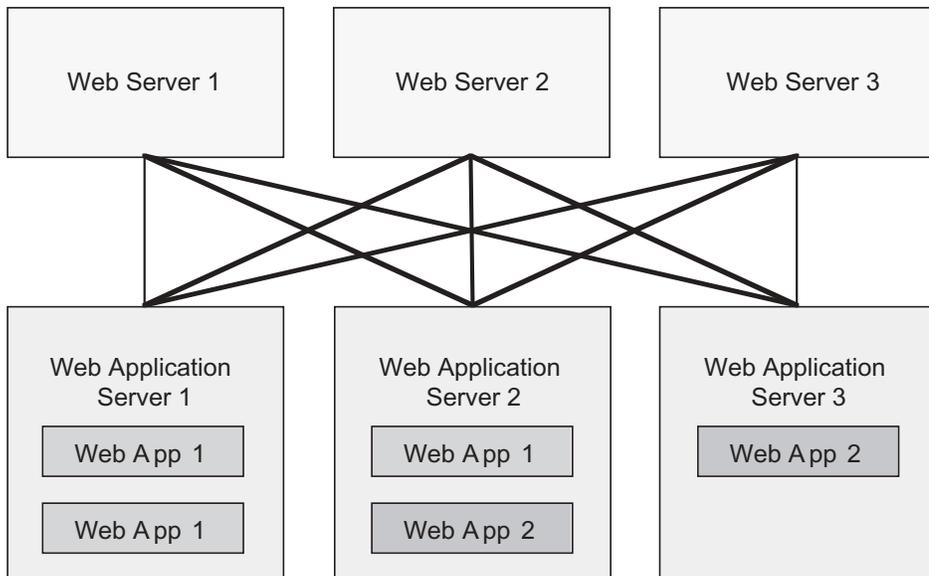
GCSS-AF treats Web applications as software components. Consequently, GCSS-AF hosted Web applications should be designed using approaches similar to the HA system design for software services. The key design elements required for Web application HA include scaling and cloning, treatment of transient data, and wise extension of user sessions with custom objects to provide a survivable storage mechanism for transient data.

Scaling and Cloning

Horizontal scaling means that a Web application has been cloned and is running on at least two independent servers configured to ensure fail-over. Horizontal cloning is required for Web applications to operate in a HA state; it provides fail-over, load balancing, additional user capacity, and enables scalability.

Assuming the application has been designed properly, cloning is a relatively simple task to perform on most Web application servers. For example, on GCSS-AF, it takes about two hours to horizontally clone a combat support mission application. Web application clones can also be placed into a vertical scaling scenario where multiple Web application instances run on one application server.

Figure 4: *Web App 1 Is Scaled Both Horizontally and Vertically. Web App 2 Is Only Scaled Horizontally. Horizontal Scaling Is Required for HA; Vertical Scaling Is Not*



Vertical scaling can provide additional capacity, and does provide server-level fail-over. Figure 4 illustrates Web applications in both horizontal and vertical scaling configurations.

Prior to establishing redundancy through horizontal scaling, a full capacity analysis should be conducted to determine the system resources required to serve the planned user base. The capacity analysis catalogs the total number of users, the average and peak concurrent user load, required application Random-Access Memory per instance, cumulative required hard disk space, and other production-related details to arrive at a physical *fielding profile* for the Web application. The fielding profile depicts how many Web application instances are necessary and lays out precisely which application servers they will reside on.

Additionally, it is important that enough capacity is provided through resource provisioning (Web and application servers, Web application instances, memory, storage, etc.) so that if any one system or application component fails, the *average* user load can still be served. In other words, extra capacity must be allocated by design to handle failures and times of peak usage. The goal of GCSS-AF is to run servers at not more than 40 percent capacity under average load conditions; this allows the necessary excess capacity.

Treatment of Transient Data

Strict separation between layers, as prescribed by the Model-View-Controller (MVC) logical design paradigm [3], results in applications that have business logic and data clearly separated (see Figure 5). This decoupling of layers greatly reduces complexity, fosters code reuse, enables flexibility, and simplifies maintainability throughout the life cycle. In short, MVC provides the necessary logical design foundation for developing maintainable and HA applications.

MVC-designed applications should not manipulate database data directly using Java Database Connectivity (JDBC); instead, they should take a pure object-oriented approach and interact with data using objects [4]. Data objects are implemented as Entity Beans or simple JavaBeans, and are buffered by the data access layer, which directly communicates with the database via JDBC.

Using the MVC logical design approach results in Web applications that are suitable for running in a HA mode because they scale nicely (both horizontally and vertically). However, using MVC

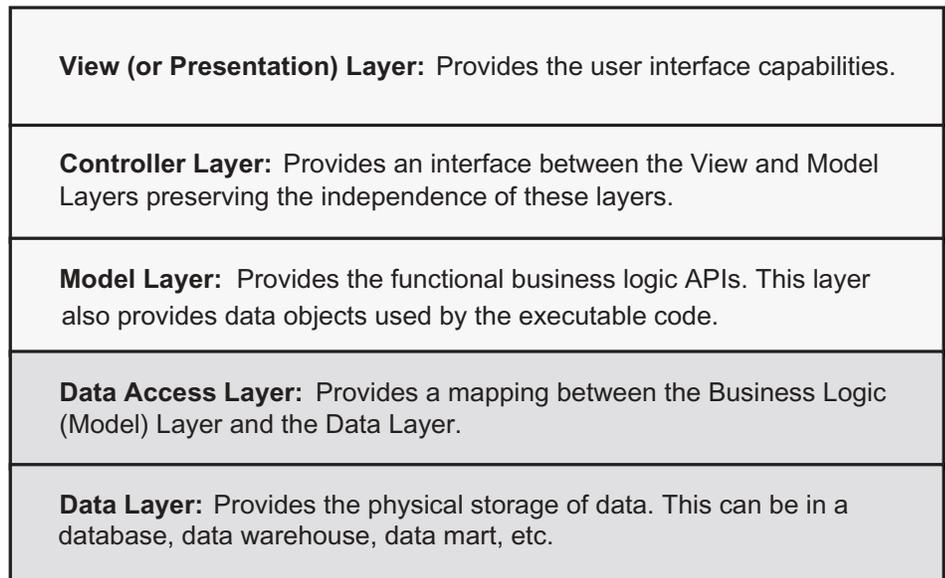


Figure 5: *Model-View-Controller Logical Tiered Architecture*

does not prevent developers from making HA design mistakes. For instance, MVC does not preclude the use of transient session data. Transient session data can best be viewed as the user's unsaved work on the Web application server. If the server fails, all transient data is lost, just like when a workstation user experiences a power outage with unsaved work. In the event of a failure, HA-enabled Web applications will fail-over to another server, however, all transient data will be lost. HA Web applications must be designed in such a fashion that a session fail-over will not result in the loss of *any* of the user's work. Specifically in Java 2 Enterprise Edition, the servlets, JavaBeans, Java classes, and Enterprise JavaBeans (EJBs) can contain transient data [5].

Session EJBs illustrate the pitfalls of using transient data in HA Web applications. Session EJBs can be either stateful or stateless. Stateful session beans maintain transient state information (in-memory data) that is used to serve or converse with the user. Stateful sessions are valuable because they allow data caching, which can dramatically improve performance. However, they do have a dark side. Should the application server fail, all of this transient data is permanently lost. For example, consider a warfighter placing an order for communications equipment (using a combat support mission Web application similar to Amazon.com). As the warfighter places items into the shopping cart, the stateful session bean keeps track of the requested items in-memory. The EJB now has state, namely this in-memory list of items. Completing the order and checking out depends on the in-memory state information remaining

available. The problem with this approach for HA Web applications is that when a server fails, this state information is lost and cannot be recovered. The user will be disrupted, and be forced to start over [6].

Properly designed HA Web applications can prevent users from experiencing transient data loss by employing stateless objects. Stateless session beans maintain no in-memory state data that could be lost in the event of a server failure. Each method in a stateless session EJB executes independently and does not rely on or store any in-memory state data. Again, consider the warfighter shopping cart example; since a stateless session bean does not maintain state data, the bean must add the warfighter's selected items to a persistent object (which moves the data into a database table via the Data Access Layer, see Figure 5). At checkout, the selected items will be retrieved from this table and the transaction will be completed. In the event of a failure, the clustered session would fail-over to another application server, and upon checkout the new application server would simply read the list of selected items from a persistent object (which pulls the data from the database table) and complete the order. Because of this resiliency, HA Web applications should only use stateless session beans. This treatment of transient data applies to servlets and regular JavaBeans as well.

Customizing the Session Object

If a Web application has the need to store information about a user in addition to what the user session object provides, a good approach is to extend the session by adding custom objects. Returning to our

COMING EVENTS

September 8-12

International Conference on Practical Testing Techniques
Minneapolis, MN

www.pstqconference.com/2003north

September 15-18

Software Development Best Practices
Boston, MA

www.sdexpo.com/2003/east

September 14-19

International Function Point Users Group Annual Conference
Scottsdale, AZ

www.ifpug.org/conferences/annual.htm

September 22-25

AUTOTESTCON 2003
Anaheim, CA

www.autotestcon.com

September 24-26

International Conference on Visual Languages and Computing
Miami, FL

www.vlc03.cs.ucla.edu

October 15-18

Richard Tapia Diversity in Computing Conference
Atlanta, GA

www.ncsa.uiuc.edu/Conferences/Tapia2003

October 20-24

Quality Assurance Joint Conference on Compressing Software Development Time
Baltimore, MD

www.qaiusa.com

November 18-21

International Conference on Software Process Improvement
Washington, DC

www.software-process-institute.com

April 19-22, 2004

Software Technology Conference 2004



Salt Lake City, UT
www.stc-online.org

warfighter shopping cart example, we could extend the user session object by adding an object called SelectedItems (which is a Java ArrayList of EquipmentItems). The warfighter's selected items can now be stored in the new SelectedItems object of the user's session. To process the checkout, these items are simply retrieved from this in-memory object (assuming memory-to-memory session clustering is used). In the event of a server failure, these objects will be treated as part of the user's session and will be subject to session clustering fail-over previously discussed. Thus, a server failure will cause a complete fail-over of the user's session and *transient data* without the user ever experiencing any interruption in service. This approach also provides a gain in performance because the SelectedItems are stored in memory instead of the database. This is a good approach for a relatively small amount of data. Large amounts of data should not be stored with the user session object [2].

Conclusion

Developing HA systems and applications is not about technology. It is about meeting the needs of the warfighter. To serve the warfighter, Web applications should be designed up-front to provide HA, which can be viewed as a composition of three components: availability, reachability, and performance. Each of these HA elements must be designed into the hosting system as well as the Web application. HA systems must ensure that the architecture *designs away* all single points of failure. The most important consideration for HA Web applications is horizontal scaling. Other important aspects of HA design include strict separation between layers (using MVC), the use of session clustering and fail-over, the treatment of transient data, and extending the session object with custom objects. ♦

References

1. Tanenbaum, A. Computer Networks. 3rd ed. New Jersey: Prentice Hall PTR, 1996.
2. IBM WebSphere Version Information Center: Session Management Support <<http://publib7b.boulder.ibm.com/wasinfo1>>.
3. Alur, D. et al. Core J2EE Patterns: Best Practices and Design Strategies. New Jersey: Prentice Hall PTR, 2001.
4. Sun Microsystems: Java Data Object API Specification <<http://java.sun.com/products/jdo>>.

5. Hall, M. Core Servlets and JavaServer Pages. New Jersey: Prentice Hall PTR, 2000.
6. Monson-Haefel, R. Enterprise Java Beans. 2nd ed. Sebastopol, CA: O'Reilly, 2000.

Additional Reading

1. Alberts, David S. Network Centric Warfare: Developing and Leveraging Information Superiority. 2nd ed revised. CCRP Publication Series, 2000.
2. Java and J2EE Timeline <<http://java.sun.com/features/2000/06/time-line.html>>.
3. Martin, J. "On Service Level Agreements for IP Networks." Proc. of the IEEE Infocom 2002 Conference <<http://www.ieeeinfocom.org/2002/papers/455.pdf>>.
4. Pressman, R. Software Engineering, A Practitioner's Approach. 4th ed. New York: McGraw-Hill, 1997.
5. Stevens, W. UNIX Network Programming. 2nd ed. New Jersey: Prentice Hall PTR, 1998.
6. Wang, Z. Internet QoS: Architectures and Mechanisms for Quality of Service. San Francisco: Morgan Kaufmann, 2001.

About the Author



Michael Acton is a software systems engineer with Lockheed Martin Mission Systems, a Capability Maturity Model® Level 5 organization. As the Global Combat Support System-Air Force (GCSS-AF) Operations and Support chief engineer, he is responsible for leading multiple engineering teams that provide the services necessary to operate the system, provide Level 2 help-desk support, install new capabilities, and integrate Java 2 Enterprise Edition-based mission applications. He was recognized as a 2002 GCSS-AF Program Top Contributor. Acton is a doctorate candidate at Auburn University.

Lockheed Martin
Mission Systems
4520 Executive Park Drive
Montgomery, AL 36116
Phone: (334) 416-6029
Fax: (334) 273-5560
E-mail: michael.acton@gunter.af.mil
michael.acton@lmco.com

© Capability Maturity Model is registered in the U.S. Patent and Trademark Office.