10TH INTERNATIONAL COMMAND AND CONTROL RESEARCH AND TECHNOLOGY SYMPOSIUM
The Future of C2

Title of Paper:
Are Service Oriented Architectures the Only Valid Architectural Approach for the Transformation to Network Centric Warfare?

Topic:
C4ISR Architecture

Author: Jack Lenahan
POC: Jack Lenahan
Organization: Office of the Chief Engineer
Space and NAVAL Warfare Systems Command
Charleston, S.C.
Address: P.O. Box 190022
N. Charleston, South Carolina: 29419
Phone: 843-218-6080
Email: John.Lenahan@Navy.mil

Abstract

The use of a Service Oriented Architecture (SOA) as the dominant single architectural design paradigm of Network Centric Warfare (NCW) introduces architectural infrastructure stability risk levels which may be unacceptable in C4ISR mission frameworks. Service reliability, service performance, and service availability represent a key set of requirements which must be satisfactorily implemented or mission risk will accelerate. The purpose of this research is to demonstrate that a complex adaptive architecture solution to the reliability, performance, and availability requirements imposed by NCW transformation provides a richer and more stable approach to DoD legacy capability exploitation than does a pure SOA approach. Using the Network Centric C4ISR Architecture Quality of Service (QoS) rating scale developed for this research paper as a behavioral classification methodology, the analysis concludes that a complex adaptive architectural model composed of at least Event Driven Components, Service Oriented Components, MOMS Components and GRID Components may present a more risk tolerant solution in terms of satisfactory implementation of NCW reliability, performance, and availability requirements. In simpler terms, a standalone SOA will be insufficient in terms of providing infrastructure stability. I propose that a highly available, disaster recoverable, GRID model (overlain with availability and performance monitoring agents) be implemented in order to sufficiently cover the reliability, performance, and availability issues needed for combat missions

Introduction

The purpose of the research is to demonstrate that a hybrid architecture solution to the reliability, performance, and availability requirements imposed by NCW transformation provides a richer and more stable approach to DoD legacy capability exploitation than does a pure SOA approach. The process followed for this research is as follows. First; define the architecture terms used (the glossary is in appendix I), second; derive a fundamental set of NCW C4ISR architecture reliability, performance, and availability requirements, third; devise a QoS & availability scale upon which architectural configurations can be classified, fourth; evaluate several common SOA models against the QoS scale, and fifth; propose possible architectural alternatives which will meet the requirements. For this paper, we are focusing only on NCW reliability, performance and availability requirements for mission support in a C4ISR combat environment.

Requirements Derivation

NCW theory postulates that service oriented style architectural paradigms will replace many platform capabilities with similar capabilities made available through “GIG Connectedness”. Implicit in this statement are many assumptions which must be translated into architectural requirements. There are three specific requirements groups which while often discussed are rarely defined: reliability, performance, and availability.

Reliability

Reliability is defined as not having a software failure which impacts the availability of any software component under use in a combat environment within the boundaries of the traditional “Mean Time Between Failure” metric. For this paper, MTBF is determined to be either empirically based, or SLA or contractually mandated. Since we are we deriving C4ISR requirements, my focus is only concerned with the MTBF which for mission critical software is defined as no failures or lack of availability during end to end mission execution periods. Traditionally, this meant that a given piece of software,

say the terrain avoidance radar or autopilot, (usually mission critical software systems come with on board duplicates or backup systems with a highly integrated failover manager) mounted within the airplane platform, had to be extremely reliable. But again, traditionally, we are only speaking about the software and its related computing environment on a platform as being required to be reliable. With the emergence of the SOA, now we are possibly looking at constructing a "Composed Terrain Avoidance Service" which the platform would subscribe to. In this case, not just the platform subscription services must be extremely reliable; the entire SOA must be highly available and therefore extremely stable. Therefore, the primary reliability requirement can be stated as follows: ***SOA software vendors must deliver software which will not fail between the start of the mission and the end of the mission regardless of mission duration.***

Availability

In order to determine if a "pure SOA" is capable of satisfying the above reliability requirement, let's further define reliability in terms of availability.

In general, mission critical SOA software must comply with the so called "5 nines" availability requirement

Five nines are derived as follows: 24 hours x 365 days year = 8760 hours * .001 = 8.76 hours per year. This translates to .024 hours daily downtime, or computed as 525600 Minutes per year * .001 = 525 minutes per year acceptable downtime, or 1.44 minutes per day unavailability. This forms the basis of the SOA and GIG Service Level Agreement and by definition; ***any component exceeding unavailability of 1.44 minutes per day violates quality of service at any level But this is in addition to and does not replace the primary availability requirement that no failures visible to the user can be tolerated during combat missions.***

Availability Discussion

Since no software is bulletproof in terms of probability of failure, I propose the following:

1. The only known industrial architectural counterpart (containing no single points of failure) is the Highly Available (HA), Disaster Recoverable (DR), and Scaleable GRID Architecture. HA means that if one service fails its clone can assume execution, this constitutes a graceful failover requirement. If the HA site fails, then the DR service clones can assume execution
2. HA and DR both require Stateful transactions at all levels of the architecture. Graceful fail over cannot occur unless there are states to be monitored and "grabbed" by the monitoring agents or fail over management software. In addition, the fail over agents must be redundant!
3. The Orchestration Engines and Choreography engines must provide HA/DR and graceful fail over and graceful degradation. Thus, IA must cover the monitoring of not only the services, their publishers, and subscribers; they must also detect and prevent denial of service attacks against the sequencing engines and also prevent intruders from re-sequencing workflows.
4. Single sign on capabilities must function in an HA DR environment.

Performance as an Availability Requirement

Is a system available if performance slips below as certain threshold? If it is too slow, it is in effect "unavailable". This is a very serious aspect of Net Centricity, since by definition users will be added sometimes in sudden spikes during surge or new conflict conditions. If it takes 30 seconds for a portlet to paint or for each data request to be serviced, in my opinion, the service was "unavailable when needed in a combat situation". Performance can degrade for many reasons. In the case of a SOA, any one of the SOA layers may be the culprit. The service itself, the orchestration or choreography tools,

the policy management layers, the single sign on or directory services, etc., all of these layers must be scaleable and perform within an acceptable boundary of degradation when high usage spikes occur. Scalability requires that given "N" number of users, the network, storage access, computer processors, and memory cannot degrade more than 10% given a "tuned" environment. For example, an acceptable standard for tuning may be that for 1 to 25 concurrent users, less than 10% performance or capacity degradation on any of the total end to end architectural elements, 26 to 50 concurrent users should experience no more than 12% total degradation.

a. If the thresholds are exceeded, route to mirrored sites – non-GRID Agent based management
b. If the thresholds are exceeded, agents select other nodes with available capacity – GRID based solution

This also has implications for the "direction" towards "pull only" SOA architectures. It may be quite necessary given a time critical ISR situation, that event publication or "pushing" is faster than "pulling". In this case, an Event Driven Architecture or sub architecture will be needed. We should not be trying to stuff an SOA into situations or activities where the requirement may call for an EDA. ***Thus, the performance requirement is that no service can degrade below a pre-defined (hopefully tagged) SLA/QoS performance threshold. Performance must be monitored and if degradation is detected, re-routing of the service must occur transparent to the user.*** The table below summarizes these requirements.

Requirement/ SOA Layer	Reliability - 99.999	Availability - HA with graceful failover – minimum N+1 architectural model	Availability as a function of performance - Max 10% degradation of SLA/QoS agreement regardless of user/usage spikes	Disaster Recovery – must be itself a "clone of the HA - Graceful failover from highly available primary systems
QoS, Security Management, and Monitoring Agents	Y	Y	Y	Y
Integration Architecture & Enterprise Service Bus	Y	Y	Y	Y
Presentation (Portal/Portlets, thin clients, etc.)	Y	Y	Y	N
Orchestration, Process, or Choreography Software Executables	Y	Y	Y	Y
Service Architecture – Includes directories (UDDIs, Services, Authentication, Single-Sign-On, Security, JCA, etc.)	Y	Y	Y	Y
Component Architecture	Y	Y	Y	Y
Service & Content Sources	Y	Y	Y	Y

Table 1 - High Level Summary of Minimum Reliability, Performance, and Availability Requirements for C4ISR SOA Architecture

Discussion of the SOA model
The model below probably depicts the early implementations of NCW. The layers provided by the article's authors show that:

Layer 1 (the source for data and potential services) is not redundant, may or may not be stateful, and has no DR capability. Thus, if a failure occurs at this level, no sophisticated restart or graceful failover is possible. DoD must decide if the initial versions of NCW will invest in the cloning of older legacy systems for HA/DR requirements satisfaction or if they are willing to accept the risk of failure which may impact a mission's success.

Layer 2 is introduced as the componentization of some subset or all of the legacy system's data and capability. This is not the SOA layer. This is an attempt at standardization of functionality in a more modular fashion such that the service or data provided by the components (old legacy system capabilities) can be accessed by a wider audience in a standard manner. Note that this graphic does not depict any vehicle to support HA/DR or scalability.

Layer 3 is the first formal SOA layer. This is where the services and data either exposed through component interfaces, or written as new web services are resident, registered, users authenticated, and made available for search engines. As the authors state, the services can exist as individual expositions or as composite web services. Again, by itself this SOA layer of UDDI, Single Sign On, Content and Service Management services are un-managed. No HA/DR or scalability exists.

Layer 4 is the business process layer of the SOA. It is at this level that both orchestration and / or choreography occur. Note that this is usually supported by commercial orchestration products such as BPEL. BPEL by itself is not HA/DR or easily scaled. Commercial products also introduce license and unique security issues which may make single sign on difficult. To remind the reader of this paper's purpose, we have not yet seen a good SOA model for HA/DR.

Layer 5 is the presentation layer. This layer is (as the author's indicate) deliberately decoupled from the SOA below it. But once again this causes issues for the SOA infrastructure architect. This is yet another layer for single sign on and for HA/DR design. How is this to be handled in the event of user surge or failures of the portal stack?

Layer 6 is the Enterprise Service Bus. This is a good method for hybrid integration, but it once again introduces single sign on complications, HA/DR and performance issues. Thus the ESB (if the SOA is implemented this manner) must be HA/DR.

Level 7 is the first level at which monitors are introduced and they are also the first peek at HA/DR, and performance management. The best architectural paradigm for this level is the Highly Available, Disaster Recoverable GRID Architecture.

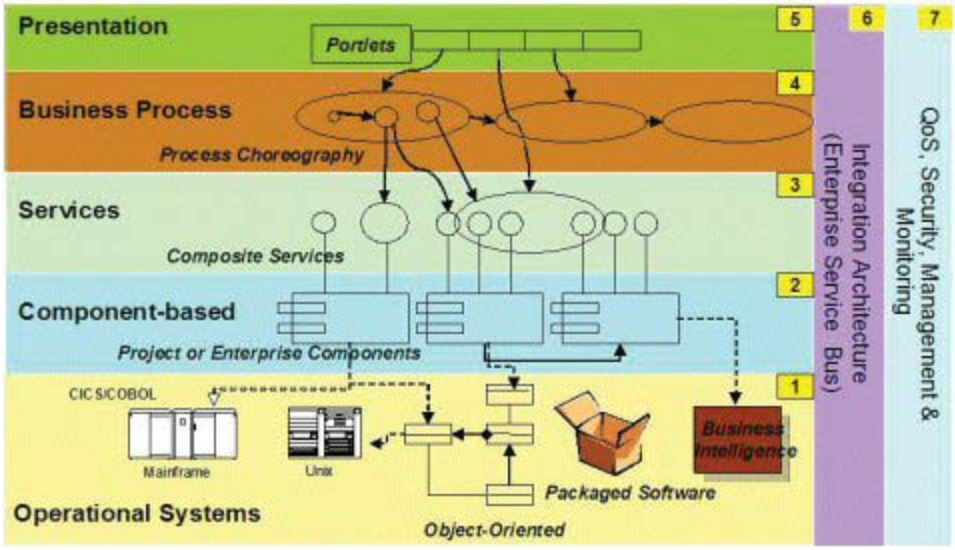

Figure 1 – Relationship of an SOA to other architectural stack layers

An extensive quotation from the authors of the figure 1 graphic and a description of this relevant research [1] follows:

Layer 1, the bottom layer, describes operational systems. This layer contains existing systems or applications, including existing CRM and ERP packaged applications, legacy applications, and "older" object-oriented system implementations, as well as business-intelligence applications. The composite layered architecture of an SOA can leverage existing systems, integrate them using service-oriented integration.

Layer 2, the component layer, used container based technologies and designs in typical component-based development.

Layer 3 provides for the mechanism to take enterprise-scale components, business unit-specific components, and in some cases project-specific components and provides services through their interfaces. The interfaces get exported out as service descriptions in this layer, where services exist in isolation or as composite services.

Level 4 is an evolution of service composition into flows or choreographies of services bundled into a flow to act as an application. These applications support specific use cases and business processes. Here, visual flow composition tools can be used for design of application flow.

Layer 5, the presentation layer is usually out of scope for an SOA. However, it is depicted because some recent standards such as Web Services for Remote Portlets version 2.0 may indeed leverage Web services at the application interface or presentation level. It is also important to note that SOA decouples the user interface from the components.

Level 6 enables the integration of services through the introduction of reliable and intelligent routing, protocol mediation, and other transformation mechanisms, often described as the enterprise service bus.

Level 7 ensures quality of service through sense-and respond mechanisms and tools that monitor the health of SOA applications, including the all-important standards implementations of WS-Management.

Lenahan Quality of Service Scale for NCW C4ISR Architectures. Now that we have established a minimum set of requirements, how can we rate whether or not a given proposed architectural solution is adequate?
An agile assessment methodology for NCW should now include "dynamic" or "near real time"assessment of the following QoS levels for any composed or pre-composed mission capability set. Individual services used individually and composed sequences (orchestrated or choreographed or both) or composite engagement packs such as the NAVY's FNEPs (FORCENET Engagement Packs are a set of pre-composed web services orchestrated and choreographed for a particular set of mission capabilities) should be granted a Lenahan QoS rating. For example, if a given orchestration of 10 web services is executing at QoS level 5 and the Orchestration tool fails at sequence number 4, then the remaining web services will never be called or activated if the orchestration tool itself is not manually restarted, gracefully failed-over, or disaster failed-over or GRID state-managed into transparent automated graceful fail over and restart. But if the sequence is executing at Lenahan level 7, then it will be automatically restarted and the remainder of the sequence will be called.

QoS Level	Capability or Capability Sets exposed as Web Services	State Recording	Simple State Recording with graceful fail over management by simple agents	Agent Monitoring of All Web Services in given C4ISR Architectural Orchestration or Choreography for Graceful Recovery of Services (Also applies to each service and its orchestration tool in a given FNEP being fully Stateful and agent monitored	HA (All enabling software / hardware infrastructure layers (Listeners, Authentication SW, Firewalls, Single Sign-on Software, Directory and Naming Management, MOMS, Database Software, Redundant Directories, Redundant data, SAN, NIC, etc) for the entire orchestration set)	HA with Full DR - Clone Of HA Suites	HA/DR with guaranteed performance management (GRIDS Only with all 7 ISO Layers HA/DR)
1	Y	N	N	N	N	N	N
2	Y	Y	N	N	N	N	N
3	Y	Y	Y	N	N	N	N
4	Y	Y	Y	Y	N	N	N
5	Y	Y	Y	Y	Y	N	N
6	Y	Y	Y	Y	Y	Y	N
7	Y	Y	Y	Y	Y	Y	Y

Table 2 – Lenahan Levels of NCW Architectural QoS for Web Services Implementations

Lenahan Levels of NCW Web Service Reliability, Availability, & Performance QoS Compliance

1. Web Services are implemented as new software or exposed existing legacy capability. No state recorded, No failover, HA, DR at this level, just the web services themselves are available, registered, discoverable, etc.
2. Web Services add State Recording (at least start or finish states with possible intermediate states recorded to a database table or a state recording service). No failover, no HA, no DR at this level. However, this positions the service for upward QoS scale movement. I realize that this is contrary to many current definitions of web services. I am recommending that we move in this direction for the achievement of greater availability, stability, and disaster recovery ability.
3. Web Service Simple State recording with graceful fail over management by simple agents. Please note that the remainder of the architecture or physical infrastructure may not include state recording, just the web services at this level. Also note that qualification at this level would not necessarily include all the web services in a given orchestration or choreography. No HA, no DR.
4. Agent Monitoring of All Web Services in given C4ISR Architectural Orchestration or Choreography for Graceful Recovery of Services (Also applies to each service and its orchestration tool in a given FNEP being fully Stateful and agent monitored). Thus, single sign-on to all services is HA, the orchestration and choreography tools are HA, and the web service monitoring agents and the web services themselves are all HA at this level). But the enabling software stacks, the portals, Apache listeners, and the enabling infrastructure do not need to be HA.
5. Complete architectural stack is Highly Available - The web services, orchestration and choreography tools, all enabling software / hardware infrastructure layers (Listeners, Authentication SW, MOMS software, Firewalls, Single Sign-on Software, Directory and Naming Management, Database Software, Redundant Directories, SAN, NIC, etc, for the entire orchestration set) – all hardware, networks, OS, and communications are automatically failed over. This means that if a legacy system is the source of the any of the services in the orchestration sequence and the legacy system is not fully HA, then the orchestration sequence defaults to a QoS level of 4. In simple terms every possible piece of software and hardware in all services in a given orchestration or choreography sequence are HA.
6. Full Disaster Recovery and first 5 nines level but without guaranteed performance management by dynamic re-allocation of compute resources or storage resources. This is level 5 plus a highly available DR clone.
7. Full Disaster Recovery and second 5 nines level but with guaranteed performance management by dynamic re-allocation of compute resources or storage resources. All layers and the orchestration / choreography tools, single sign-on, and the orchestrated web services and their source computer sets fully HA/DR. Performance is not impacting availability.

General discussion

Let’s compose a C4ISR mission sequence for a simple problem: search and rescue.
First we must get a request to go look for a downed helicopter.
Second, we decide that we must launch a drone with video to look for the helicopter.

Third, neural net pattern recognition software is assigned the first analysis task of evaluating the streaming video.
Fourth, but concurrent with the intelligent aid, a human is tasked to view the video
Fifth, upon potential helicopter identification, the human analyst tasks a small recovery team to the correct location.

If we were to choreograph this search and rescue task sequence using the SOA model above, we have a task probability of software failure at each layer. Each layer has its own failure probability based upon the number of composite services, unique services, components, message delivery mechanisms, and legacy functions and interfaces. A failure anywhere in any layer kills the particular choreography step or workflow if you prefer. However, a failure of the choreography engine kills everything. A failure of Single Sign On kills everything. The "pure SOA" if you will is inadequate to provide any meaningful avoidance of terminal single points of failure. By using the Lenahan scale, and including the composition's Lenahan score in a Meta tag, a pre-simulation QoS factor is immediately known. Risk assessment can be assisted using this simple technique.
By using the Lenahan QoS scale and interrogating the QOS tags, we can now know the following for an individual service or a composition:

1. Are there SOA single points of failure?
2. Are there component layer single points of failure?
3. Are there single sign on or security or identity management single points of failure?
4. Are there data or service source provider (legacy e.g.) single points of failure?
5. Is there networking, messaging, or communications single points of failure?
6. Are there orchestration or choreography single points of failure?
7. Does the composition have at least total HA (graceful failover) at all layers?
8. Does the composition have at least total HA/DR (graceful failover) at all layers?
9. Do all layers utilize performance monitoring agents in GRID environment?

Analysis – Guiding Requirement – Neither poor service performance nor single points of failure can be tolerated
Now let's examine these requirements at some level of detail, and propose a possible model capable of delivering such an ambitious set of reliability and availability requirements. The first level of analysis indicates that for mission critical services during combat operations, the service must be available, 24hours 7 days a week for the duration of the engagement or mission. This means that any interruption of service due to non recoverable failures at any level of the SOA is intolerable. Thus, all networks, communications, computers, services, orchestration packages, single sign on, and other SOA software at all levels must provide for graceful failover and non-interrupted services. Since the primary data transfer in many cases of an SOA will be messages, it is worthwhile to examine at this point in the paper, relevant research contributed towards the goal of reliable messaging [2]. Since SOAs and web services themselves are relatively new to the software field, a certain level of distrust in terms of reliability is natural. However, there is hope in the empirical data provided by users of a Message Oriented Middleware Architecture (MOMS) approach. The following is a long quote from the above citation: "A Messaging Alternative". "Message-oriented Middleware is already deployed in many mission-critical systems at enterprise sites around the globe and has proven to be a valuable enhancement to enterprise architectures providing scalability and reliability in easily manageable product suites. In short MOM is a mature technology. It is still gathering momentum and may prove with time to be the natural partner for Web services initiatives, together providing a symbiotic nervous

system for mission-critical data within and without the firewall. …. This maturity can be utilised by the Web services community to provide higher QoS levels than are possible through HTTP communication methods alone and in turn HTTP can open the conduit to allow the flow of data to permeate the firewall in a bi-directional manner. Rather than rely on the HTTP protocol and our web server of choice to be the backbone of our multi-billion dollar enterprise solution, architects can make use of the advantages which MOM affords, including:

- Reliability - messages are recoverable should there be a problem
- Scalability - MOM has already proven itself in high-throughput projects
- Performance - once the message has been placed on a queue, it is behind the firewall and can be treated as any other JMS (Java Message Service) TextMessage object, no proprietary (de)serialization is required
- Return On Investment - middleware does not come cheap and integration projects can be expensive, by reusing existing messaging capabilities heterogeneous data exchange becomes less expensive, less time-consuming, and less disruptive
- Leveraging existing skillsets - MOM can be a highly specialised area and a manager wants to make the most of their people investment, it is not necessary to pay for expensive consultancy to integrate your applications
- Fast, Guaranteed delivery - algorithmic logic built-into most MOM products will ensure priority messages take the quickest route and no message will ever get 'lost'
- Asynchronous capability - Publish-Subscribe breaks out of the bi-directional (HTTP) paradigm and allows messages to be sent back to one or many registered client applications at some point in the future
- Transactions and failbacks - HTTP is stateless, MOM will provide transactional support for the payload and return an application to its previous state in the case of an error “

To continue the original thread now, since failures are inherent to the nature of software, especially new software, as will be the case with the new NCW capability deployments, state of execution information must be maintained for each layer. Once state has been recorded it must be monitored. The monitors cannot fail or we have simply moved the single point of failure, thus the state monitors must be monitored, in commercial architectures, this is known as a highly available (HA) design. Figure 2 below, depicts the above model in an HA configuration.

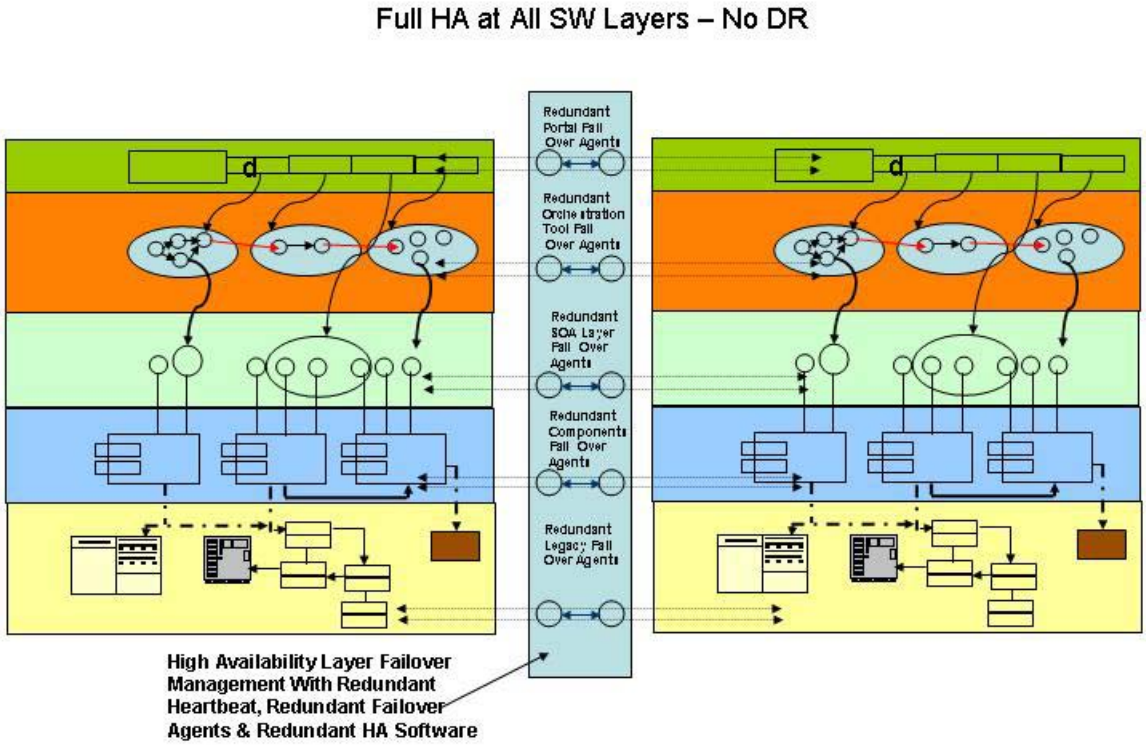

Figure 2 – All Software layers of the Architecture are HA

Since we are speaking of five nines class availability, the next level of analysis would seem to indicate that the HA system itself must be able to gracefully recover from a disaster (the total loss of the primary HA systems), thus a complete clone of the HA system will be needed. This clone is known as the highly available disaster recovery system, also HA, since if a disaster occurs, then the failover target must be highly available itself. Figure 3 below depicts the HA/DR concept.

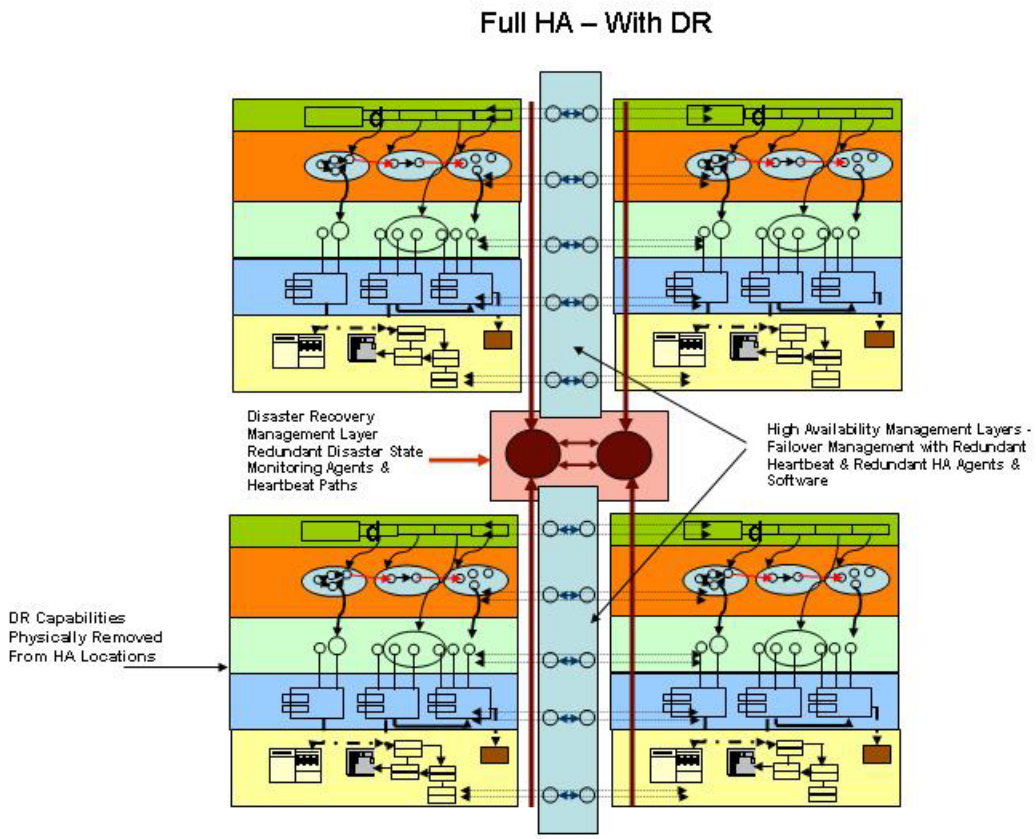

Figure 3 – All Software Layers of the Architecture are HA & DR

Commercial industry stops at this level of redundancy as the standard definition of five nines, thus, so will the author accept this depth of backup as a boundary. Now that we have covered the HA and the DR, we still have one more requirement set to cover, performance due to surges in users or workload due to a new conflict. The only commercial model currently available to even attempt dynamic routing of service executables due to performance is the architecture known as a GRID. The GRID architecture depicted below would be overlain with the software layers depicted in figure 3 above. The GRID is designed to manage both HA, DR, and re-allocate software due to poor SLA/QoS. If a service is not meeting he required SLA, GRID performance agents can move the software to a freer node in the GRID. Indeed, the GRID scheduler probably will schedule services for execution using computation resources with the lightest workloads in the first place in order to preemptively manage future computational or other resource bottlenecks. The formal definition[3] that I am using is as follows:

A GRID is an "IT infrastructure that (1) supports dynamic resource allocation in accordance with service-level agreement policies, efficient sharing and reuse of IT infrastructure at high utilization levels, and distributed security from edge of network to application and data servers and (2) delivers consistent response times and high levels of availability—which in turn drives a need for end-to-end performance monitoring and real-time reconfiguration." Figure 4 below depicts a GRID.

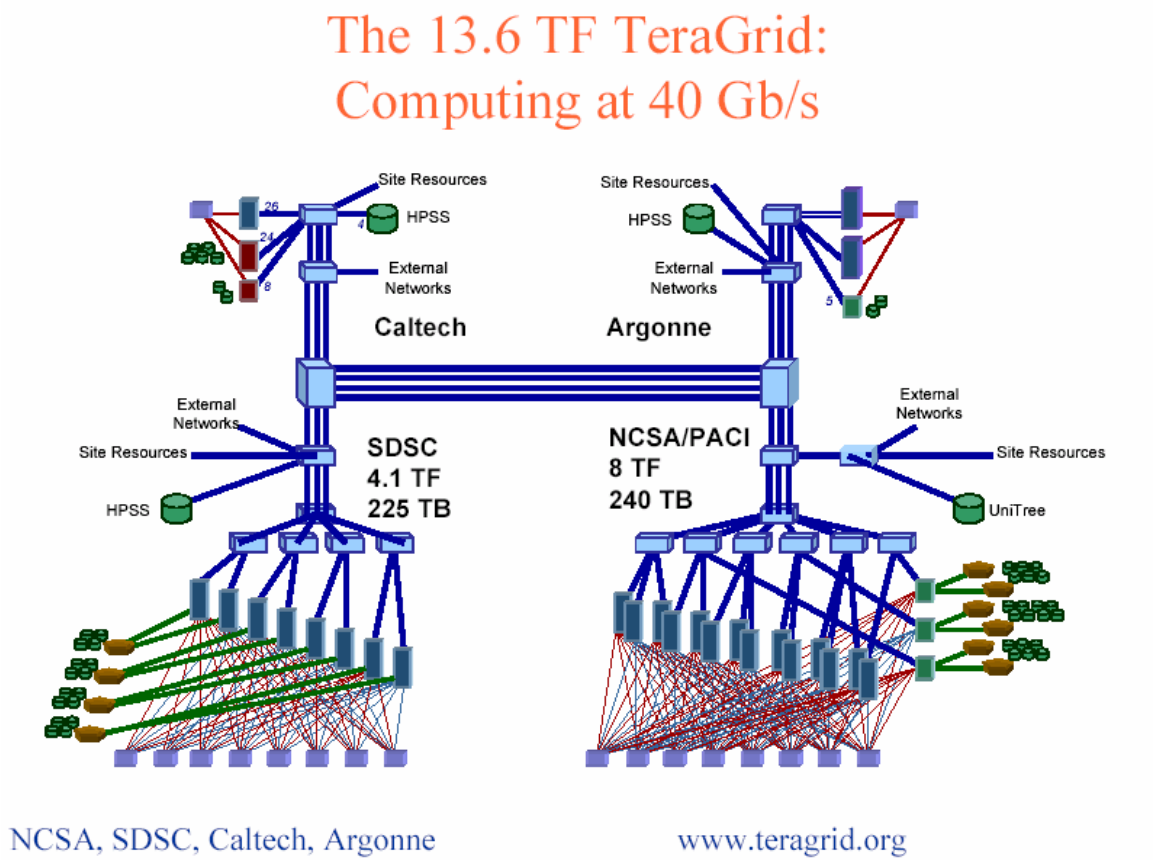

Figure 4 – GRID[4] Architecture – A Complex Adaptive System which is scaleable, highly available, disaster recoverable, and capable of dynamic program execution resource re-assignment.

Please note that the diagram depicts the paper's intentions with respect to HA and DR. Thus, in this diagram, the left side contains redundant physical networks, computers, and storage, if the networks, storage and computers each contain failover and heartbeat monitors, then the left side is totally infrastructure HA. By adding the SOA software layers redundantly, then the left side is SOA and Infrastructure HA. The fact that this is a GRID permits dynamic resource reallocation and scheduling if a particular web service begins to experience poor or degraded performance. Thus the architecture "adapts" to failure as well as degrading performance. Performance monitoring agents can have degradation thresholds set at any arbitrary level to immediately attempt to re-allocate the degrading services executables to a more adequate resource set upon threshold crossing detection. The existence of the right side's infrastructure at a physically different location (CALTECH vs. Argonne) qualifies this instantiation as disaster recoverable.
The fact that this architectural model can adjust to individual component failures as well as poor performance makes it easily fit into the definition of a Complex Adaptive Architecture.

Conclusion: a standalone SOA will be insufficient in terms of providing infrastructure stability. I am proposing that a highly available, disaster recoverable, GRID model (overlain with availability and performance monitoring agents) be implemented in order to sufficiently cover the reliability, performance, and availability issues needed for combat missions.

A GRID infrastructure, with HA/DR monitoring of all components including the services and their sources themselves, should be selected to achieve this level of quality and availability.

Summary of Analysis shows the following:

1. **That all levels of the model, including the communications and networking not depicted, must be highly available to support 5 nines availability, (one set of clones) and disaster recoverable (a second set of clones)**
2. **That all tools embedded in the SOA (particularly the choreography, orchestration, and single-sign-on software) must also be redundant**
3. **The HA/DR monitoring agents themselves must be HA/DR**
4. **Increased use due to new conflicts or surge deployments must not introduce degraded performance. This requirement almost by itself should be enough to justify the expense of a full GRID architecture as the underlying infrastructure of the SOA. We should not assume that an SOA will be performance scaleable in mission critical environments without a GRID.**
5. **No single points of failure can be tolerated. Simply stated, a break in any software component at any level will cause the service to be unavailable if HA/DR technologies are not implemented.**

References and Other Relevant Research

1. Delving into Service-Oriented Architecture, By Bernhard Borges, Kerrie Holley and Ali Arsanjani -
   September 17, 2004 (Please note that I have both included the original and modified the original, the publisher of the cited material has requested that I post the following legal information: basically that no commercial implementations or sales of the material can be pursued with their permission. WWW.Developer.com/design/article.php/3409221 - **http://www.jupitermedia.com/corporate/legal.html - Jupiter Media Notice 1. Copyright, Licenses** MAY NOT MODIFY, COPY, REPRODUCE, REPUBLISH, UPLOAD, POST, TRANSMIT, OR DISTRIBUTE, IN ANY MANNER, THE MATERIAL ON THE SITE, INCLUDING TEXT, GRAPHICS, CODE AND/OR SOFTWARE. Subject to more specific terms on individual JUPM web sites, you may print and download portions of material from the different areas of the Site solely for your own non-commercial use provided that you agree not to change or delete any copyright or proprietary notices from the materials (certain areas require paid license fee prior to downloading any material). You agree to grant to JUPM a non-exclusive, royalty-free, worldwide, sub licensable, perpetual license, with the right to sub-license, to reproduce, distribute, transmit, create derivative works of, publicly display and publicly perform any materials and other information (including, without limitation, ideas

contained therein for new or improved products and services) you submit to any public areas of the Site (such as bulletin boards, forums and newsgroups) or by e-mail to JUPM by all means and in any media now known or hereafter developed. You also grant to JUPM the right to use your name in connection with the submitted materials and other information as well as in connection with all advertising, marketing and promotional material related thereto. You agree that you shall have no recourse against JUPM for any alleged or actual infringement or misappropriation of any proprietary right in your communications to JUPM. **and Idea Submissions.** Domestic and International copyright and trademark laws protect the entire contents of the Site. The owners of the intellectual property, copyrights and trademarks are JUPM, its affiliates or other third party licensors.

2. Building Enterprise-class Web Services using Messaging-oriented Middleware, Author: Chris McGarel (Senior Software Engineer, Cape Clear Software)
3. Source: The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration , Ian Foster , Carl Kesselman , Jeffrey M. Nick, Steven Tuecke Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 2 Department of Computer Science, University of Chicago, Chicago, IL 60637 3 Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292 4 IBM Corporation, Poughkeepsie, NY 12601
4. "Grids and High Performance Distributed Computing" - Andrew Chien , March 31, 2004, CSE225, Spring 2004 - The TeraGrid project is funded by the National Science Foundation and includes nine partners: NCSA, SDSC, Argonne, CACR, PSC, ORNL, Purdue, Indiana, and TACC. Any questions or comments please email the webmaster@teragrid.org.

Appendix I – Glossary of Architectural Terms Used in This Research

Definition of Terms – These definitions apply for this paper. It is not the intention of the author to debate alternative definitions. That debate is still ongoing and quite frankly, contributes to the confusion surrounding the intent of the NCOW and NCW literature.

Availability – the binary presence (yes/no) of an architectural component when needed by a user of the component. If a component has low reliability (fails often or in excess of a contractual SLA/QoS agreement set), then it will probably not be available when needed or during the entire duration of the mission. High availability usually means that a redundant component is made present and that both the original and clone components monitor each other or are monitored by some agent to provide as transparent as possible non-interruption of the service. (Author's Definition)

Choreography

A choreography defines the sequence and conditions under which multiple cooperating independent Web services exchange information in order to achieve some useful function.

Source: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis

Client Server

Client Server - a server which processes a request and returns the results to the requesting computer
Source: www.jegsworks.com/Lessons/reference/glossary-c.htm

Client/Server distributes the processing of a Computer Application between two computers the Client & the Server - the principal being to exploit the power of each. The Client is normally a PC. The Application Program will access data and perform processing on the Server and using the data obtained via the server more processing tasks will be performed on the Client. More than one user can use the application.
Source: www.cvc2.org/survival_guide/web/web20a.htm

A relationship between programs running on separate machines in a computer network. The server is the provider of services, while the client is the consumer of the services.
Source: www.dartmouth-research.com/glossary_internet_PR.html

Event Driven Architecture –
An Event Driven Architecture is a software architecture which is centered on the state changes (events) of an automata or database. The EDA is characterized by the initiation of message traffic due to the occurrence of change of a database row or rows through insertion, deletion, or updating. In the case of automaton state changes, events are usually described as program subroutines completing, object oriented methods completing, or the initiation of a request for data by a user. Errors and keyboard activities are also considered events which can trigger message traffic. The occurrence of the event usually results in the transmission of data in the form of a message, the primary direction of message flow is considered as a "push" rather than a "pull". (Author's definition)

EDA – from www.webopedia.com
Short for event-driven architecture, an enterprise software infrastructure model in which events trigger the real-time exchange of messages between independent software applications. EDA relies on an event-processing agent that detects events across an enterprise and, using a push approach, notifies all of the other software applications that need to be notified of the change in data, all at the same time. For example: the e-commerce Web site of an enterprise receives an order for a product, completing a business event. An event agent detects this transaction and simultaneously notifies all other applications in the enterprise that need to know about the order, which can include such aspects as an inventory database, accounts receivable software, customer service applications, marketing and advertising monitors, and shipping software.

This method is different than more traditional methods of enterprise communication in which events are compiled in batches and then communicated across the enterprise at periodic intervals.

Failover

Definitions of failover on the Web:

When one individual computer fails, another automatically takes over its request load. The transition is invisible to the user.
iishelp.web.cern.ch/IISHelp/iis/htm/core/iigloss.htm

The transfer of operation from a failed component (e.g., controller, disk drive) to a similar, redundant component to ensure uninterrupted data flow and operability.
www.jmr.com/support/glossary.html

The transfer of operation from a failed component (e.g., controller, disk drive) to a similar, redundant component to ensure uninterrupted data flow and operability.
www.sunrise.uk.com/glossary.html

A fault-tolerant clustering architecture in which two servers share a common set of fault-tolerant fixed disk drives. In the event of failure of one of the servers, the other transparently assumes all server processing operations. See clustering and fault tolerance.
Source: www.pace.ch/cours/glossary.htm

A function to substitute a failed system component for a redundant component. In Multi Path Driver, if the current path fails, I/O is rerouted through a redundant path so that the system can continue production operations. See also 'Multi Path Driver'.
storage-system.fujitsu.com/global/term/

– In the event of a component failure, its function is automatically assumed by a redundant component.
www.iomega.com/europe/support/english/documents/11240e.html

Graceful Failover

Graceful (no-data-loss) failover - A no-data-loss failover is possible if the corresponding primary database is operating in either the maximum protection or maximum availability data protection mode.

…. a **switchover** operation that is a graceful role reversal between the primary database and one of its standby databases. Role management services also minimize downtime from an unplanned failure of the primary database by facilitating the quick *fail over* to one of the standby databases through a **graceful failover** or **forced failover** operation.

Source: Oracle9i Data Guard Concepts and Administration Release 2 (9.2)

GRID – IT infrastructure that (1) supports dynamic resource allocation in accordance with service-level agreement policies, efficient sharing and reuse of IT infrastructure at high utilization levels, and distributed security from edge of network to application and data servers and (2) delivers consistent response times and high levels of availability—which in turn drives a need for end-to-end performance monitoring and real-time reconfiguration. Source: The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration Ian Foster1,2 Carl Kesselman3 Jeffrey M. Nick4 Steven Tuecke1 1 Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 2 Department of Computer Science, University of Chicago, Chicago, IL 60637 3

Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292 4 IBM Corporation, Poughkeepsie, NY 12601

Mainframe - A term originally referring to the cabinet containing the central processor unit or "main frame" of a room-filling Stone Age batch machine. After the emergence of smaller "minicomputer" designs in the early 1970s, the traditional big iron machines were described as "mainframe computers" and eventually just as mainframes. The term carries the connotation of a machine designed for batch rather than interactive use, though possibly with an interactive time-sharing operating system retrofitted onto it; it is especially used of machines built by IBM, Unisys and the other great dinosaurs surviving from computing's Stone Age.
It has been common wisdom among hackers since the late 1980s that the mainframe architectural tradition is essentially dead (outside of the tiny market for number crunching supercomputers (see Cray)), having been swamped by the recent huge advances in integrated circuit technology and low-cost personal computing. As of 1993, corporate America is just beginning to figure this out - the wave of failures, takeovers, and mergers among traditional mainframe makers have certainly provided sufficient omens (see dinosaurs mating). Supporters claim that mainframes still house 90% of the data major businesses rely on for mission-critical applications, attributing this to their superior erformance, reliability, scalability, and security compared to microprocessors.

Source: *The Free On-line Dictionary of Computing, © 1993-2004 Denis Howe*

Portal - An integrated and personalized web-based interface to information, applications and collaborative services. Access to most portals is limited to corporate employees (an intra-company portal) or corporate employees and certain qualified vendors, contractors, customers and other parties within the extended enterprise (an inter-company portal).
Source: www.upstreamcio.com/glossary.asp. Compare this to a grand and imposing entrance (often extended metaphorically); "the portals of the cathedral"; "the portals of heaven"; "the portals of success"
Source: www.cogsci.princeton.edu/cgi-bin/webwn

Portlet - Portlet refers to pluggable web components that process requests and generate content within the context of a portal
Source: http://docs.sun.com/source/816-6758-10/ch6.html

Reliability
Reliability is defined as not having software failure which impacts the availability of any software component under use in a combat environment within the boundaries of the traditional “Mean Time Between Failures” and Mean Time To Repair. For this paper, MTBF is used and determined to be either empirically based or SLA or contractually mandated. Since we are we deriving requirements, my focus is only concerned with the MTBF which for mission critical software is defined as no failures or lack of availability during end to end mission execution. Thus, for a 3hour mission, an SLA must state and the vendor must deliver software which will not fail between the start of the mission or the end of the mission. Traditionally, this meant that a given piece of software, say the terrain avoidance radar or autopilot,(usually mission critical software systems come with on board duplicates or backup systems with a highly integrated failover manager) mounted within the airplane platform

had to be extremely reliable. But there we are only speaking about the software and its related computing environment on a platform as being required to be reliable, with the emergence of the SOA, now we are possibly looking at constructing a "Composed Terrain Avoidance Service" which the platform would subscribe to. In this case, not just the platform subscription services must be extremely reliable; the entire SOA must be highly available. (Author's definition)

Reliability – extremely low mttf/mtbr for an architectural component (hw/sw)

Single Sign On – The ability to login once to a GRID and access any capability or application without additional logins (author's Definition)

Service, A service is a self-contained, stateless function which accepts a request(s) and returns a response(s) through a well-defined interface. Services can also perform discrete units of work such as editing and processing a transaction. Services are not dependent on the state of other functions or processes. The technology used to provide the service is not part of this definition.
Source: Wikkipedia

SOA – Service Oriented Architecture – A software architecture which is layered upon an existing physical infrastructure, and also layered upon existing enabling software infrastructure, which delivers content and services to consumers primarily via the use of Web Services. Service Oriented Designs permit the simplification of content delivery services which can be used as components or "building blocks" by C4ISR mission architects. (Author's definition)
From web – (Service Oriented Architecture) - A system for linking resources on demand. In an SOA, resources are made available to other participants in the network as independent services that are accessed in a standardized way. This provides for more flexible loose coupling of resources than in traditional systems architectures.
Source - http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

State – the existential condition, attributes, configuration, or status of a system, device, of software component. Simple examples usually refer to a switch as being in one of two possible conditions: on or off. Thus, a switch has an "on state or an off state". (Author's definition)

Stateless, Not depending on any pre-existing condition. In a SOA, services are not dependent on the condition of any other service. They receive all information needed to provide a response from the request. Because services are stateless, they can be sequenced (orchestrated) into numerous sequences (sometimes referred to as pipelines) to perform business logic.
Source: Wikkipedia

Web Services discussions that are listed below are all sourced from W3C and thus grouped together for cohesion of explanation

What is a Web service?

There are many things that might be called "Web services" in the world at large. However, for the purpose of this Working Group and this architecture, and without prejudice toward other definitions, we will use the following definition:

[Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]

Source: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis
Agents and Services

A Web service is viewed as an abstract notion that must be implemented by a concrete agent. (See Figure 1.) The agent is the concrete entity (a piece of software) that sends and receives messages, while the service is the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular Web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language). Although the agent may have changed, the Web service remains the same.

Source: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis

Service Oriented Architecture

The Web architecture and the Web Services Architecture (WSA) are instances of a Service Oriented Architecture (SOA). To understand how they relate to each other and to closely related technologies such as CORBA, it may be useful to look up yet another level and note that SOA is in turn a type of distributed system. A distributed system, consists of discrete software agents that must work together to implement some intended functionality. Furthermore, the agents in a distibuted system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks that are intrinsically less reliable than direct code invocation and shared memory. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) consider the unpredictable latency of remote access, and take into account issues of concurrency and the possibility of partial failure. [Samuel C. Kendall, Jim Waldo, Ann Wollrath and Geoff Wyant, "A Note On Distributed Computing"].

An SOA is a specific type of distributed system in which the agents are "services". For the purposes of this document, a service is a software agent that performs some well-defined operation (i.e., "provides a service") and can be invoked outside of the context of a larger application. That is, while a service might be implemented by exposing a feature of a larger application (e.g., the purchase order processing capability of an enterprise resource planning system might be exposed as a discrete service), the users of that server need be concerned only with the interface description of the service. Furthermore, most definitions of SOA stress that "services" have a network-addressable interface and communicate via standard protocols and data formats.

Source: Source: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis

SOA and REST Architectures

The World Wide Web is a SOA that operates as a networked information system that imposes some additional constraints: Agents identify objects in the system, called "resources," with Uniform Resource Identifiers (URIs). Agents represent, describe, and communicate resource state via "representations" of the resource in a variety of widely-understood data formats (e.g. XML, HTML, CSS, JPEG, PNG). Agents exchange representations via protocols that use URIs to identify and directly or indirectly address the agents and resources. [Web Arch]

An even more constrained architectural style for reliable Web applications known as "Representation State Transfer" or REST has been proposed by Roy Fielding and has inspired both the TAG's Architecture document and many who see it as a model for how to build Web services [Fielding]. The REST Web is the subset of the WWW in which agents are constrained to, amongst other things, expose and use services via uniform interface semantics, manipulate resources only by the exchange of "representations", and thus use "hypermedia as the engine of application state."

The scope of "Web services" as that term is used by this Working Group is somewhat different. It encompasses not only the Web and REST Web services whose purpose is to create, retrieve, update, and delete information resources but extends the scope to consider services that perform an arbitrarily complex set of operations on resources that may not be "on the Web." Although the distrinctions here are murky and controversial, a "Web service" invocation may lead to services being performed by people, physical objects being moved around (e.g. books delivered).

We can identify two major classes of "Web services":

- REST-compliant or "direct resource manipulation" services in which in which the primary purpose of the service is to manipulate XML representations of Web resources using the a minimal, uniform set of operations operations,
- "distributed object" or "Web-mediated operation" services in which the primary purpose of the service is to perform an arbitrarily complex set of operations on resources that may not be "on the Web", and the XML messages contain the data needed to invoke those operations.

In other words, "direct" services are implemented by Web servers that manipulate data directly, and "mediated" services are external code resources that are invoked via messages to Web servers.

Source: Source: http://www.w3.org/TR/2003/WD-ws-arch-20030808/#whatis

The Service Oriented Model

The Service Oriented Model focuses on those aspects of the architecture that relate to Service and action.

The primary purpose of the SOM is to explicate the relationships between an agent, the services it offers and requests.

While it is clearly the case that an agent cannot offer or request a service without being able to send and receive messages, the SOM does not mention messages or message transport. The SOM builds on the MOM; but its focus is on action rather than message.