The Intelligent Architectures Design Philosophy

By John S. Howard - Enterprise Engineering

Sun BluePrints™ OnLine - December 2001

http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 816-3653-10
Revision 1.0, 11/13/01
Edition: December 2001

The Intelligent Architectures Design Philosophy

IS/IT staff are continually faced with the need to architect systems, whether for the deployment of new systems and services or in the redesign of existing systems and services. Sun's Enterprise Engineering group has developed *Intelligent Architectures* as a technique to architect systems and datacenters.

This paper will introduce the design philosophy and tenets of the Intelligent Architectures (IA) approach to systems architecture; a philosophy centered on the use of archetypes—original models after which similar things are patterned.

This paper will present the IA archetypes in brief, as well as rules and recommendations for combining archetypes to architect systems and datacenters. Additionally, recommendations for the combination of archetypes through the use of traditional design techniques such as top-down or bottom-up and structured or ad-hoc design will be explored.

Throughout this paper the terms "application" and "service" are used interchangeably to refer to a software construct that provides a function to other systems or end-users. The term "server" is used to refer to the hardware platform upon which a service executes. Finally, the term "system" is used to denote the combination of a server and all of the applications provided by that server.

Quality of Design

The IA approach is built on several tenets that we consider to be essential to "good design." However, before examining these tenets the concept of what is meant by a "good design" should be clarified.

For our purposes, a good design is one that helps maximize the reliability, availability and serviceability of a system. Further, a good design is as simple as possible while providing maximum extensibility. This simplicity enables the system to be easily understood, enhancing the maintainability of the system.

A good design must allow for the inevitable failure of a component or of the system as a whole. A good design provides recovery tools that aid in the rapid recovery of the system.

Finally, a good design must meet the requirements for the system, regardless of whether all of the requirements are known or whether they are accurate. It is only through an agreed upon set of requirements that a solution can be formed and eventually judged a success or failure. In the absence of known requirements, assumptions must be documented, and, most importantly, agreed upon with the customer (or end-user). The requirements definition process, like the design process, can be viewed as an iterative process that produces successive refinement.

Intelligent Architectures Design Tenets

Clustering for highly-available (HA) services, for example, with Sun™ Cluster software, is an area where service requirements must be examined very carefully. When implementing an HA service, there is a common misconception that the boot disk does not need to be mirrored because the service can be failed-over to another host in the event of a boot disk failure. While this is true, it does not account for the time the service is unavailable during the fail-over process. Although it may not take the fail-over very long to complete, if the requirements of the service are for five nines (99.999%) of availability, you should avoid the fail-over time altogether by mirroring the boot disk.

The inverse of this concept is also interesting to consider; it may not always be necessary to mirror the boot disk on systems providing an HA service. Consider the HA system management services provided by System Service Processor (SSP) to the Sun Enterprise™ 10000 platform; all Sun Enterprise 10000 configurations should have two physically distinct SSPs; a main SSP, and a spare SSP. With this configuration, there is no need to use a logical volume manager (LVM) to mirror the system disk of the main or spare SSP. In the event of a system disk failure in the main SSP, the SSP services can simply be failed-over to the spare SSP. The unavailability of the SSP during fail-over will not affect any running domains on the platform managed by the SSP. Further, because the SSP has only one system board and one SCSI disk controller (which are both single-points of-failure), the complete physical redundancy of SSPs gives you greater protection than mirroring with an LVM on the main or spare SSP.

In short, no single rule or best practice can be given that is appropriate for all systems in all datacenters. While you should take the datacenter site standards and experience of the datacenter personnel into account, the requirements and needs of the application must be the driving force for the decisions and compromises made in designing a system and planning its boot disk.

The IA design tenets are:

- Design for the needs of the application
- Employ an iterative design process
- Design with simplicity in mind
- Employ reusable design components

The following sections explain these tenets.

Design For the Needs of the Application

It is crucial that the development or choice of software tool be based on the requirements and needs of the system or datacenter. Consider the need of choosing an LVM, such as the Solstice DiskSuite™ software or VERITAS Volume Manager (VxVM). All too often, the choice of LVM is based on emotion, uninformed opinion, misunderstanding, or misconception. While the system administrator's experience and comfort level with an LVM are important, these factors should contribute to the decision of which LVM to implement, but must not be the driving force in the choice.

Before creating or implementing a system architecture, you must define and understand the availability and serviceability requirements of an application or service. These requirements must be the driving force in selecting and implementing a system and system software. The availability and serviceability needs of the application are paramount.

Further, the availability and serviceability requirements of an application must be addressed in the design of a system. For example, a system to be used exclusively to provide a data warehouse service is ideally suited to have its database implemented on a RAID5 volume, especially RAID5 implemented in hardware of the storage device or enclosure. The data warehouse transaction mix, which is almost entirely database reads, is well suited to RAID5 and data redundancy, and availability is achieved without the high cost of availability that RAID1+0 would impose. The read-oriented nature of the data warehouse allows the major weakness of RAID5 (extremely slow writes) to be avoided.

It is important to note that the preceding example did not mention the operating system (OS) or boot disk. The OS and the on-disk image provided by the boot disk exist to provide an environment for the application or service to function. The system architecture, of which the boot disk is a key component, must be designed and implemented to optimize the application, not the OS. In the preceding example,

the LVM that provides the most efficient implementation of software RAID5, or the LVM that works best with the hardware RAID5 subsystem of the data warehouse, is the best LVM for the system.

This concept of system architecture is the architecture side of the axiom used in performance tuning; That axiom states that the greatest performance increases are gained in tuning the application, and the least performance increases are achieved in tuning the OS.

Employ an Iterative Design Process

When creating a system architecture, be aware that very few (if any) systems are deployed and then put into stasis. Systems, by definition and by necessity, are in an almost constant state of flux. The system life cycle is used to describe the phases a system or component goes through. These phases usually include initial concept, deployment, sustaining maintenance, and retirement. With many systems or components in a modern IT datacenter, the system life cycle dictates that constant changes will occur.

This concept of a system life cycle maintains that nearly every system will be designed and then redesigned several times in its lifetime. In fact, the analysis of existing system architectures and system redesign, or remodeling, is quite possibly one of the most common activities for system administrators and IT architects. They are rarely given an empty datacenter, a multibillion dollar budget, and told to “create a datacenter.” More often, system architects are given a datacenter with legacy systems, very little remaining floor space, aggressive schedules, and insufficient budget.

The main constraint when remodeling is that one can’t start over from scratch. Further, constraints like cost, risk, time, fallback plans, and return on investment create additional problems to remodeling systems. Making appropriate choices regarding the system architecture are critical to being able to work within these constraints.

Simplicity in Design

In addition to masking potential reliability issues, system complexity is often one of the largest inhibitors to system recovery. Because less-complex systems are more readily and quickly understood (and, therefore, easier to troubleshoot) than complex systems, decreased system complexity may help speed system recovery in the event of a failure.

Simplicity in design also benefits junior system administrators and junior datacenter operations staff by enabling them to administer systems that otherwise may have been beyond their experience or abilities.

Additionally, decreased system complexity helps minimize potential exposure to software bugs or harmful interactions between software components. Simply by decreasing the number of installed software components, you can reduce the potential for bugs or harmful interactions between software.

Reusability of Design Components

Reusing system components offers many advantages to the system architect. By having a common and reusable component design, the time required to architect systems decreases.

Additionally, reusable components provide a “known quantity” to system designs. The software components can be debugged and their correctness verified before integration into the system architecture. By combining these reusable components, you can design and deploy systems faster than you could by designing each system from the ground up.

Further, utilizing reusable components increases consistency across all systems in the datacenter. This minimizes the “one-off solutions” and ensures adherence to site standards.

This consistency across systems is, quite possibly, the greatest benefit of using reusable design components. Consistency across systems in your datacenter improves system recovery, simplifies maintenance procedures and run books, and enables systems to be installed and deployed faster.

The benefits and importance of reusable components are so great that the IA design philosophy is built around the utilization of *archetypes*, an original model after which similar things are patterned; essentially reusable components.

The Intelligent Architectures Archetypes in Detail

The IA archetypes can be considered reusable components, however, they are not “objects” or “black boxes” that can be dropped into place and used as-is. Archetypes take into account the hard datacenter reality that only the most trivial of problems can be resolved by an unmodifiable black box. Archetypes can be viewed as lumps of clay rather than pythagorean solids.

Archetypes are abstractions of hardware and software services, defined to meet the most basic system architecture needs. More importantly, archetypes are also defined to be adaptable and extensible to the *real-world* needs of a datacenter. To achieve this, archetypes have attributes to augment or modify the behavior or function of an archetype. To use an analogy from the physical sciences, the archetype may be thought of as a molecule, and its attributes as the atoms that comprise the molecule.

For example, all archetypes have a security attribute. This security attribute is used to control and modify all security and access aspects of the archetype. In the case of the data store archetype, the security attribute defines the security policies and file access permissions for the data to be placed in the data store.

By designing systems from the reusable, well-defined, debugged archetypes, consistent systems can be readily deployed and managed in the datacenter.

The following sections describe the IA archetypes and the common archetype attributes.

Data Store

The Data Store archetype abstracts all data storage resources including memory (RAM and virtual memory), disk drives, storage area networks (SAN), and tape drives. It is important to note that not all of the sub-components of the Data Store archetype are persistent memory. For example, server memory (RAM) is included as part of a data store. A data store also includes services such as logical volume management to combine attributes or sub-components of the data store to provide reliable and highly available data stores.

The Data Store archetype serves as a repository for all storage resources, and the needs of the application determine the appropriate resource to be used from the repository. For example, if the application requires reliable, available, persistent storage, RAID1+0 volumes would be used.

Data Manipulation

The Data Manipulation archetype is the repository for resources and services that manipulate or manage data. For example, this archetype abstracts the CPU and computing resources, clustering, fail-over, and highly available (HA) frameworks.

For the purposes of systems architecture, the operating system is considered data. This implies that the management and layout of the on-disk operating system image is considered part of the Data Manipulation archetype.

Data Transport

The Data Transport archetype is the abstraction for moving data between the Data Store and the Data Manipulation archetypes. This can include system buses, as well as networks and network protocols.

User Interface

The User Interface archetype serves as the repository for the requirements and specifications for how data is presented to the end user, as well as the requirements and methods of access.

It is important to note that in this sense, the user may be another system or service. This concept is crucial for combining archetypes into systems and combining systems into datacenters.

Physical Plant

The Physical Plant archetype is comprised of the physical requirements and needs of a system. Examples include power requirements, optimal operating temperature range, and required floor space.

The Physical Plant archetype also contains location information for the datacenters and systems. This includes the location of the datacenter and building that house the system or archetype, as well as information about its location within the datacenter.

Nerve Center

The Nerve Center archetype is the abstraction for the control and maintenance functions necessary for a system. This includes datacenter and system management and notification applications (such as Sun™ Management Center software), systems administration, service level agreements, datacenter and operations staff, and staging environments to perform unit and integration testing.

Business Continuity Planning

The Business Continuity Planning (BCP) archetype is the repository for the requirements and components for providing uninterrupted services during business disruptions or natural disasters.

This archetype provides the disaster recovery plans, as well as service requirements during a disruption.

Life Cycle

The Life Cycle archetype is the abstraction for the management and planning of the design, deployment, maintenance, and eventual decommission of a system.

This archetype embodies the procedures and processes of deploying a system and validating that it is production-ready, as well as maintenance information and runbooks for a system.

Archetype Attributes

As previously mentioned, archetypes are augmented or modified by the use of attributes.

The functional areas that the attributes address are:

- Security
- Persistence of data
- Reliability
- Availability
- Serviceability
- Simplicity over complexity
- Capacity
- Monitoring and measurement
- Process, procedure, change management, training, documentation, etc.
- Backup
- Recovery
- Commonality over diversity

These attributes will affect the archetypes, the exact manner in which the attributes affect or modify the archetypes is dependant on the archetype. For example, the monitoring and measurement attribute of the Data Store archetype is used for monitoring storage utilization and data access times. However, monitoring and measurement of the Physical Plant archetype is used for monitoring the datacenter environmentals such as ambient temperature, humidity, quality of power, etc.

It is important to note that reliability, availability, and serviceability are three separate attributes. While these attributes are typically considered one concept, they are three distinct attributes that may often be in contention with each other. For example, increasing the availability of an archetype may have a detrimental effect on the serviceability of the same archetype.

Consider the case of mirroring several disks with a Sun StorEdge™ D1000 array enclosure. The Sun StorEdge D1000 array enclosure has redundant power supplies, multiple SCSI controllers, and the bus can be split into two electrically distinct buses. However, the Sun StorEdge D1000 array enclosure has a common backplane for both buses. This backplane is not an availability issue or single-point-of-failure (SPOF), as its failure will not affect both halves. However, as the entire Sun StorEdge D1000 enclosure must be powered down to replace this backplane, it does provide a serviceability issue.

This is not a flaw in the Sun StorEdge D1000 array enclosure; it is a design point that must be considered. Only the needs of your application can determine if the Sun StorEdge D1000 array enclosure is sufficient.

Note – these attributes can be used to help enforce best practices and consistency across the archetypes. For example, if government requirements necessitate that data be retained for 10 years, the backup and recovery attributes can implement the requirement with the Data Store and Life Cycle archetypes of the pertinent systems.

More importantly, the archetype attributes are a way to document and analyze the trade-offs made when designing a system. Consider the case of a business need that dictates the creation a “one off” system to resolve a specific problem or meet a goal. For example, the site standard may be to use Sun Cluster 3.0 software to provide highly available NFS service for user’s home directories and development workshops. However, business needs dictate the use of an old version of the source code sharing and management software for maintenance of a certain software project. This requirement, in turn, forces the use of Sun Cluster 2.2 software instead of the Sun Cluster 3.0 software.

This divergence from site standards is a trade-off that is made to meet the business need for maintenance of the software project. The commonality attribute of the affected archetypes can be used to document the reasons for this trade-off, as well as the impact of the trade-off on other archetypes or systems.

Note that some attributes may be detailed requirements, such as reliability, availability, or backup. Other attributes, such as simplicity and commonality, are more of a reminder to the architect, and are to be used to document any divergence from site standards.

Combining Archetypes Into Designs

The power of archetypes comes from their ease of combination to form systems. For example, a database system is designed by combining data store (persistent, reliable volumes and the access and capacity requirements of the database) with data manipulation (the CPU requirements most likely expressed in terms of maximum query and lookup times) and data transport (the network requirements of the database). The User Interface archetype specifies the requirements, and explains how users of the service access the database. Further, the Business Continuity Planning archetype specifies the requirements for disaster recovery and the requirements and methods for providing uninterrupted (or scaled-down) services during prolonged events such as a flood, fire, power outage, etc.

As it may have been apparent in the discussion of archetypes, there is natural combination, and in some cases a mutual recursion or self-reference, between some archetypes. For example, the Data Store archetype may contain a SAN. The SAN is implemented using fibre channel disks, and the fibre channel disks communicate on the fibre loop using a network protocol. This fibre loop and network protocol is also a part of the Data Transport archetype.

This duality is intuitively understood, and taken advantage of, by experienced system architects. This duality is in part due to the perspective that a system architect uses to approach a problem. To a network-oriented system architect, a SAN is a network of disk drives. To a storage-oriented system architect a SAN is physically dispersed set of disk drives, that communicate with each other.

A system architect's perspective is greatly influenced by their experience. It is this experience that allows the system architect to resolve the duality of the archetypes and know when it is beneficial or appropriate to treat a SAN as a component of the Data Transport archetype, and when to treat it as a component of the Data Store archetype.

Structured Design vs. Ad Hoc Design

Structured design refers to identifying and understanding all of the factors that affect a design, as well as all requirements, prior to design.

While ad hoc design is structured, its structure isn't planned in advance. To take an example from outside the IT industry, most cities in the world are the result of ad hoc design. In most cases, cities developed over time without a single plan defining the entire city. It must be stressed that ad hoc design is not chaos and it should not

be used as an excuse to “just let things happen.” As with the example of the growth of a city, buildings are not built haphazardly, and most, if not all, cities have zoning and building requirements.

It may be obvious that a structured design is a near impossibility in the real world. Rarely are system architects presented with a complete set of detailed system requirements from which to design a system. Especially in the case of a system redesign or remodeling, a system architect is given a rough and incomplete set of requirements and a set of constraints to work within.

As with the experienced system architects approach to combining archetypes and resolving the dual nature of some archetypes, the experienced system architect will employ a dualistic design methodology.

System architect’s experience will guide them through concurrently applying both a structured approach and an ad hoc approach. The structured approach will be used in those areas where the requirements or implementation issues are well-defined or well- understood. When confronted with a problem area that is poorly defined or without hard requirements, system architects will switch to an ad hoc approach. This helps enable system architects to “fill in the gaps” in the requirements or problems definition.

As with many things, this ability, or the knowledge required to determine when to apply it’s use, comes only from attempting to architect systems and from making mistakes.

The concept of switching between these two design approaches is easily understood. However, knowing when to switch from one process to the other during the design process, as well as knowing what requirements are essential, comes only with practice.

To illustrate this, consider the need to architect a highly available mail and messaging server based on the Solaris™ 8 Operating Environment and iPlanet™ Message Server 5.1 software.

The crucial areas of this problem area require you to consider the following:

- What are the availability requirements of this service?
- What are the availability expectations of the users of this service? (“Do I need to meet the expectations of my users or do I need to provide 24x7 service?”)
- What, if any, service or response time guarantees have been made?
- What are the users privacy expectations, what privacy guarantees have been made or implied, and how secure must the service and data be?
- What, if any, are the backup, data restore, and data retention needs? (This will help determine the backup schedule, restore requirements, and business continuity planning).
- On what scale is this to be implemented (hundreds of users, thousands of users, millions?)

The previous questions serve to determine the “hard requirements” that must be met. These hard requirements define the structure which can then be filled in by choosing components from the archetypes.

The hard requirements will help determine if an HA framework, such as Sun Cluster 3.0 software, is necessary to meet the availability requirements. Hard requirements will also help determine the attributes that the archetypes must take, as well as any required inter-relationship between those archetypes. For example, by questioning the number of users and required response time, you can determine the capacity attribute of the Data Store and the Data Manipulation archetypes.

After these design points have been addressed, a more ad hoc approach can be used to fill out the rest of the design. These ad hoc design items include determining the process and documentation attributes, identifying serviceability and recovery attributes (which can also be used to provide a run book for the service), and specifying the criteria for the monitoring and measurement attributes.

Summary

This paper has introduced the basic concepts of the IA design philosophy and has introduced the design philosophy and tenets of the IA approach to systems architecture. The following IA tenets were presented: designing the system to the needs of the application, employing an iterative design process, striving for simplicity, and using archetypes; reusable system components.

This paper also presented the IA archetypes, as well as recommended combining archetypes to architect systems and datacenters. Additionally, recommendations for the combination of archetypes through the use of traditional design techniques (such as top-down or bottom-up and structured or ad-hoc design) were explored. Further, the duality of archetypes, as well as the concurrent application of design methods was examined.

Finally, an example of how to approach and analyze a system design task using the IA philosophy was presented.

Acknowledgements

The ideas presented in this paper grew out of the work of Richard Elling, Mark Garner, and John S. Howard.

Thanks to Julie Snow for applying her skill and expertise to this paper. And, as always, thanks to Chuck Alexander and Bill Sprouse (The Management) for the constructive criticism, guidance, and the latitude to do what we do.

Author's Bio: John S. Howard

John S. Howard has over 19 years experience in software engineering and systems administration. As a Senior Staff Engineer in the Enterprise Engineering group at Sun Microsystems he is currently working on projects for enhancing system availability and serviceability. Prior to his position in Enterprise Engineering, John worked as an Area System Support Engineer with Sun Enterprise Services. As an ASSE, he was responsible for providing escalation management and backline system support for problem isolation and resolution in the areas of clustered systems, the storage subsystem and the Solaris kernel. In addition to these support functions, he developed and performed Reliability, Accessibility, and Serviceability (RAS) studies of customer datacenters.