# THE OPERATIONAL POINT OF VIEW

## ARCHITECTURAL RECOMMENDATIONS FOR INFRASTRUCTURE SOFTWARE

PRINTED: FRIDAY, AUGUST 29, 2003

PREPARED BY HILBERT COMPUTING, INC.

# TABLE OF CONTENTS

**OVERVIEW**......2

**OPERATIONAL SOFTWARE**......3

**Logging Services**......3

**Metrics Services**......5

**Debugging Services**......6

**Security Services**......6

**COMMON APPLICATION SERVICES**......8

**A FRAMEWORKS-BASED APPROACH**......9

**CONCLUSION**......12

# OVERVIEW

No one shopping for a home ever remarked on how nicely the concrete for the basement foundation was poured. Any homeowner that has dealt with a leaky basement has an appreciation of the value of infrastructure done right and the problems that arise when it is not. Such is the plight of infrastructure software.

Infrastructure software is code that is applicable across application domains and addresses two broad areas:

- **Operational software**. This covers the facilities needed to produce diagnostic information on how the applications are running. It also includes access to security services.

- **Common application services**. This includes common base classes and frameworks that can be used across applications in a variety of domains. The effective use of common services can reduce the total amount of code that needs to be developed. It can also increase the stability of applications since there is more mature, presumably debugged, code than if each application were developed from scratch.

Infrastructure software differentiates itself from other software in that the customer for both types of software are IT professionals instead of personnel working on the core business. Application developers are the customers for common application services. The operational staffs are the customers for the operational software.

The focus and consequently the funding for applications development tend to be on the business logic. This is reasonable and expected, but there needs to be a focus on the needs of operational staff that have to support the application after it has been placed in production.

This will document the recommendations and architectural considerations for operational software. In addition, some discussion will be made on recommendations for common application services.

# OPERATIONAL SOFTWARE

Once software has been deployed in a production environment, it is the responsibility of the operational staff to ensure that the application remains available for customers. If there is dysfunction within the software, the operational staff is the first line support to gather diagnostic information and try to make a determination on the source of the problem.

Ideally, the operational staff is to be able to gather sufficient documentation at the first failure such that the problem can be resolved before the problem occurs again. It is also desirable if the operational support organizations are aware that there is a problem at the same time or before the customer so that outages can be kept to a minimum. It is these lofty ideals that drive the recommendations for the set of operational software.

The operational software architecture should be a part of the overall application architecture. An ideal application architecture is implemented using a frameworks-based approach to development. The framework itself can provide a minimal amount of operational information with no coding effort on the part of the applications developers. This concept is covered in more detail later in this document.

## LOGGING SERVICES

The first operational facility that should be included in all applications is a logging facility. Logging facilities report events within an application ranging in severity from informational messages to reporting severe unrecoverable situations. When there is an application failure, the logs provide a time line of when things were last working properly and hopefully information about the event that caused the software failure.

The primary design criterion for logging facilities is that the application programmer's interface into the logging facilities should be decoupled from the disposition of the logging information. That is, the application should be able to record significant events without knowing what happens to the information once it is recorded. The operational staff then has the flexibility to manage the log information in a way that best serves the needs of the operational environment. This abstraction between the application recording the information and the management of the recorded information gives the operational staff to implement a variety of ways of managing the information as the infrastructure changes, including the integration with operational services such as those from CA-Unicenter or Tivoli.

For Java development, there are two well-architected solutions for logging information:

- **Jakarta Log4J**. This is an no-cost, open-source logging facility that was originally developed by IBM and donated to the Apache Jakarta project.

- **JDK Logging**. Starting with JDK v1.4, Sun Microsystems included a standard logging facility.

In addition to these two widely-used logging facilities, there is a logging facility that predates both that was included in the no-cost, open-source frameworks from Hilbert Computing. The recommendation is that both Log4J and JDK logging[1] be supported for Sprint-developed applications and that the Hilbert logging be supported as a legacy only.

All three logging facilities support the architectural notion of separating the disposition of the logging information from the application programmer's view of logging. Log4J and JDK logging both allow the programmer to specify the handler for the logging information either programmatically or declaratively in properties files. In all three circumstances, application's developers should ensure that the handlers are specified declaratively so the operational staff has the flexibility to determine the disposition of the log information.

## DISPOSITION OF LOG INFORMATION

There are a specific set of needs that need to be met by the backend system that manage the disposition of the logging information:

- The logging information from multiple application servers and applications need to be aggregated. The WebLogic servlet containers have multiple instances in a cluster. It is helpful to see all the logging information for the cluster as a whole.

- The logging information must be available remotely. The Unix servers on which the servlet containers run are not generally accessible by the programming staff. It would be beneficial for applications developers to have access to the log data without a need to logon to the Unix servers.

- The log information should have the ability to be filtered by application or component so that administrative staff would have the ability to focus on the information for a single application.

1 This is not available until JDK 1.4. As of 22-Aug-2003, the WebLogic release at Sprint only supports JDK 1.3.x. In the short term, all logging should be done via Log4J.

- From an architectural point-of-view, it is desirable for the disposition of the logging information to be in a separate process[2] so that the management of logging information isn't in the code path of the running application.

Note that most of these requirements are met by the current logging infrastructure in use by some of the servlets and applications supported by the WEBS group. While it is recommended that the application programming interface be the standard JDK logging or Apache Log4J, the disposition of the logging information can be handled by the Log Monitor application currently in place. It is recommended that the Log Monitor application be used since this currently meets the needs of the operational groups at Sprint.

## METRICS SERVICES

Metering an application provides low-level operational data that can be very useful for subsequent operational analysis. Metering is simply counting and timing activities within the application. For example, servlet HTTP requests or SQL transactions times.

Most applications developers understand the need for logging and will put some level of diagnostic reporting within their applications. That is typically not the case for metering. A very small subset of applications developers are aware of the operational needs for counters and timers within their applications. For this reason, the overall application architecture should provide some minimal level of metering in the base level of code so that *all* applications can participate in SLAs. The best approach for implementing metering for all applications is for the application architecture to use a frameworks-based approach and for the framework to provide metering services in the base code. This topic is discussed in more detail later in the document.

There are three primary areas in which the operational data can be put to use:

- **Capacity Planning**. Data can be collected and analyzed for capacity planning and performance monitoring activities. A combination of profiling information and usage information (“hits”) can be used to create a statistical model that can assist in projecting the capacity requirements as an application is rolled out to a larger base of customers.

2 aka “address space” or in the case of Java, a Java Virtual Machine

- **Performance changes**. Similarly, if customers are reporting response time issues, the historical timing information for an application can be used to determine if the application has slowed between application releases and/or with usage increases.

- **Service Level Agreements** (SLAs). Selected metering information can be used to determine if the applications and operational staffs are meeting service level agreements. It is relatively straightforward to meter if a certain percentage of HTTP requests are being serviced with a certain amount of time or if SQL transaction are completed within a certain amount of time.

Like logging, the application programmer's code to initiate metering information should be abstracted from the disposition of the metrics information. This will enable the operational staffs to change out backend systems for storing and processing the information without affecting the application.

A similar design and architecture that exists for the logging framework should be in place for the metrics framework. That is, the best implementation is for a separate process to handle the disposition of the metering information. That will remove the operational processing from the code path for processing applications requests, thus helping to maintain a responsive application. Having a separate, networked process means that the metrics data can be processed on a separate server if the need should arise.

## DEBUGGING SERVICES

Logging provides ongoing operational information that can be used to verify the operation of an application and provide diagnostics in the event of failure. Occasionally, more detailed information is desired in order to diagnose more complex problems.

Debugging services are differentiated from logging services in that they are typically not enabled during the normal operation of the applications.

## SECURITY SERVICES

A very important piece of operational software provides authentication and authorization services in order to protect corporate information assets. In addition, the software should provide auditing to track failed access attempts so that appropriate action can be taken.

The ideal implementation of security services would provide a mechanism that will create an abstraction layer between a segment of code that needs to be protected and the concrete implementation of the authentication. This will enable vendor-supplied authentication services to be replaced without affecting the application as the security requirements chance.

It is vital that the authentication be handled via security services that are largely external to the application. That is, the techniques for authentication are abstracted in such a way that the application gets authenticated credentials, but the authentication technique is external to the application. In Unix systems, this is implemented using the Pluggable Authentication Module (PAM) architecture, which is mirrored in the Java 2 Authentication and Authorization Services (JAAS) security model. By delegating the authentication to external security services, corporate-wide changes from a userid/password system to a future system, perhaps based on PKI[3], biometrics or split-password technologies such as SecureID can be accomplished with a minimal amount of disruption to the application.

Authorization – basically the set of permissions associated with an authenticated subject – should also be collected and assigned by operational code outside of the application. Under no circumstances should application code directly populate authorization from the security store through SQL or LDAP calls. The level of abstraction from the application code should be such that the security store can be replaced as the operational needs change.

By abstracting and externalizing the security services, the operational and administrative staffs can move toward a single security store that is managed corporate-wide instead of on an application-by-application basis. This dramatically reduces the administrative costs of security and improves the security by minimizing the security changes required when an employee leaves the company or transfers to another part of the organization.

More detail on the techniques for implementing pluggable security can be found the in the separate document entitled "*Pluggable Security, Techniques for Object-Level Security in Java*".

3 PKI stands for "Public Key Infrastructure", which is typically implemented with X.509 certificates

# COMMON APPLICATION SERVICES

The primary focus of the document is operational software. However, there is considerable opportunity in providing common application services. This simply uses industry best-practices to provide facilities that a lot of applications will find useful.

A considerable amount of high-quality code is available through the Apache Jakarta Project. In particular, the Struts framework deserves considerable attention. It significantly reduces the time to construct servlets. The architecture provides facilities for assembling reusable application components outside of code.

When common application services are obtained from a vendor or the open-source community, the operational aspects of these code bases should be reviewed as well as the functionality with respect to applications developers. In many cases, the base classes need to be extended to provide operational support, typically for diagnostics and metering. Such was the case with the Struts framework. While it is an excellent presentation layer framework, the diagnostics were severely lacking in version 1 of the product. The base classes were extended to provide diagnostics and metrics. It can be expected that this will be a requirement for other class libraries used within the Sprint infrastructure.

In addition to the facilities from the Jakarta Project, frameworks exist for scheduling the execution of code at some future time and XML processing.

# A FRAMEWORKS-BASED APPROACH

Regardless of the software products chosen for Sprint-developed applications, it is highly recommended that a frameworks-based approach to development be taken to implement all applications.

Frameworks-based development leverages object-orientation to create more flexible software systems. Frameworks are characterized by the heavy use of design patterns, particularly the factory design pattern, so that object-oriented components can be assembled and coupled to a base set of software facilities. The use of factories to construct and assemble software components accomplishes two goals. It creates a structural basis for components so that reuse can be accomplished by pushing functionality into base classes or creating reuse by making services available in the framework into which the components are assembled. The second goal is that a framework tends to narrow the possible software designs from "anything that we can code" to a structural basis that becomes easier to extend and maintain than recreating software from the ground up each time software functionality is needed. An effective framework provides valuable software services without constraining the developer too much in their goal of solving the problem at hand.

If implemented correctly, frameworks also provide a valuable aid in providing diagnostic information to the operational staffs. As one would expect, applications developers are focused more on delivering application functionality than providing operational support information. If applications are built on a framework, the framework itself can provide a considerable amount of operational information without the developer having to explicitly write code.

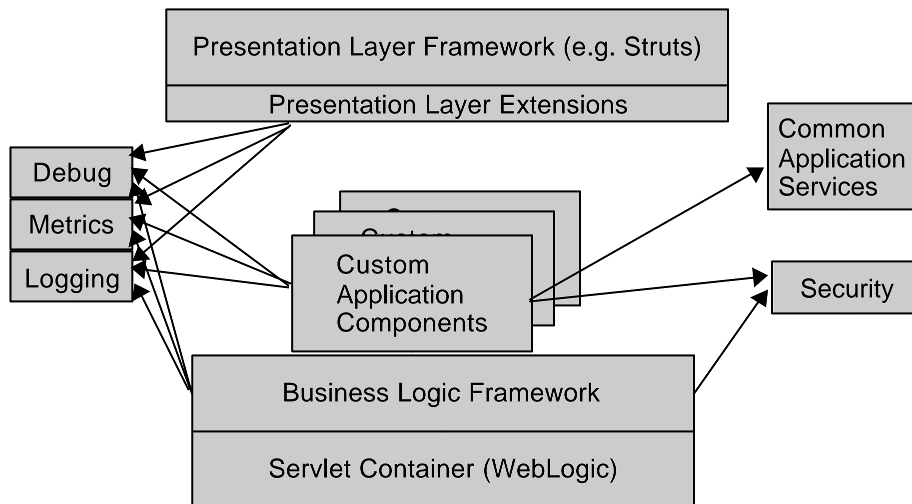
There are two real-world examples where the effective use of frameworks became invaluable:

1. One company's servlets were developed using extensions to Struts to assemble both the presentation layer and the business logic components. After the code was developed and in production, security calls for authorization were placed in the base part of the framework. The framework for business logic components was extended to support declarative security. The result was that security was added to all servlets developed for this customer *without adding or changing a line of existing code*, except to add a logon JSP to collect userid and password information and authenticate the credentials entered.

2. A thin applications framework was developed with code stubs, but no implementations for several operational services such as metering and logging. Applications were developed to the framework level. Independent efforts were placed on the development of the operational code, such as managing the disposition of the logs. The first applications were deployed in production with little operational diagnostics.

   Because the application framework provided an abstraction layer between the applications and the operational services, when the operational code was ready for production, the new code was implemented and instantly, all applications were imbued with logging, metering, etc. with no changes to the code.

The framework layers are shown in the following diagram:



Using the diagram as a visualization aid, the following points can be made about the value of a frameworks-based approach to applications development:

- The series of custom application components is meant to indicate that applications are comprised of isolated units of code that are assembled by the business logic framework in a manner external to the code. Typically, this is accomplished through an XML description document that identifies the components to be assembled along with their configuration values.

- The business logic framework provides calls to the operational services available to all applications. The implication is that all applications will have a minimal amount of operational diagnostics and metering capability with no application developer effort.

- Of particular note is the application framework calls to the security services. Because the custom application components are described in a declarative XML description document, their security requirements can also be declared as part of the descriptor. Since the business logic framework is used by all applications, all applications can be endowed with declarative security as granular as the application components with no additional programming.

  Note the custom application components can also make security calls if more granular programmatic security is required, but the expectation is that relatively few applications would require explicit security calls.

- The presentation layer extensions were developed to accommodate a lack of diagnostic capability within the base Struts framework from Apache's Jakarta project. Some base classes were subclassed so that additional logging and debugging facilities could be added.

# CONCLUSION

The ability to support applications after they are deployed in a production environment is crucial to provide maximum availability at the lowest possible operational cost. The structure for a rich operational environment is documented here.

While the operational services are important in achieving the goal of supportable applications, a frameworks-based approach to applications development is equally important. Since operational considerations are not always paramount to developers focused on delivering business functionality, it is vital that a core set of operational information is provided "for free" to applications developers. In particular, the ability to add a considerable level of support for authorization without coding security calls within an application is a compelling reason in itself.

As a final note, all of the architecture concepts presented in this document are available via open source, royalty-free software. While some of the features are in production at Sprint, *all* of these features are in production in one or more customers of Hilbert Computing.

**Author Information**
*Gary Murphy*
*Hilbert Computing, Inc.*
*13632 S. Sycamore Dr.*
*Olathe, KS 66062*

*e-mail: glm@hilbertinc.com*
*voice: (913) 780-5051*