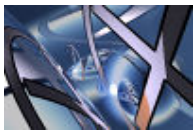
Article

A RESTful ESB implemented using NetKernel



Posted by **Jeremy Deane** on Apr 08, 2008 12:15 PM

Community Java Topics Web Services Tags NetKernel

Background

A top-tier New England University has adopted a strategic multi-year infrastructure modernization project to replace out-of-date systems, and increase IT capabilities while maximizing the return on all IT investments. The project involves upgrading hardware, purchasing new software and training the development and operational groups. Central to the modernization strategy is the implementation of a Service Oriented Architecture (SOA).

RelatedVendorContent

A Technical Introduction to Terracotta

Hibernate without Database Bottlenecks

IBM software architect eKit: Grady Booch podcast, whitepapers, articles

RESTful todo list sample tutorial with Groovy & Project Zero

IBM Web 2.0 Developer eKit: Free Tutorials, Webcasts, Whitepapers

SOA is an architectural approach to development that emphasizes an overall platform for the design of distributed applications rather than specific technologies. The mainstay of SOA is the definition and implementation of software services, regardless of location or ownership, that map directly to a system or business processes-services include interfaces and the policies that govern them at run-time. SOA benefits include loose coupling between interacting systems and platforms, a ubiquitous integration mechanism based on industry standards, support for on-demand creation of composite services and the ability to leverage existing assets while improving operational efficiency.

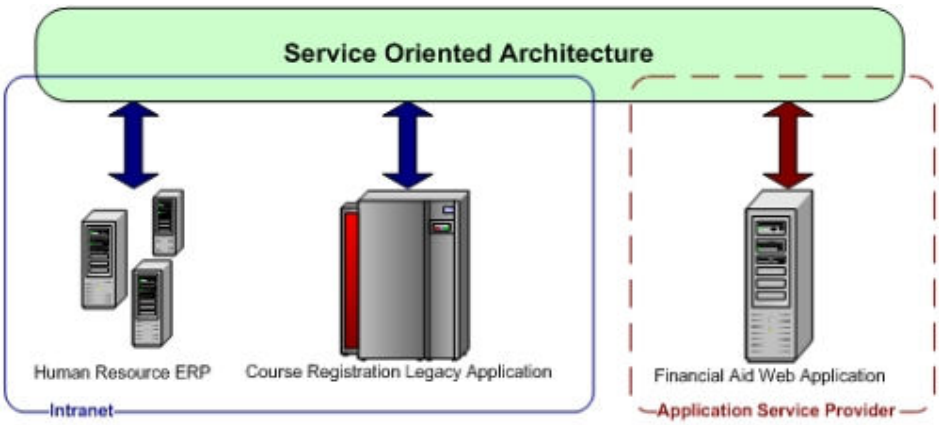


Figure 1 Service Oriented Architecture

The transition from traditional application development and deployment to a service-based approach is significant and cannot be put into practice overnight. Working with their partner, Collaborative Consulting, the University's IT department outlined a roadmap for incremental adoption of SOA. One advantage of the incremental approach is an immediate return on investment and the ability to select the ordering of conversions to best meet the short and long terms goals of the University. The remainder of this article describes a six-month project that launched the SOA adoption process by implementing a *Resource Oriented* Enterprise Service Bus (ESB) using 1060 Research's NetKernel.

Problem Domain

Higher educational institutions are under constant pressure from students, staff and alumni to provide the type of on-line services they have grown accustomed to in their lives including real-time 24/7 access to information via intuitive interfaces and streamlined automated processes. At the same time there is increasing pressure from management, steering committees, and boards to control costs. Consequently, higher educational institutions IT departments such as the University's must be ingenious and pragmatic when investing in new capabilities.

The University IT department supports a wide variety of applications including Commercial off the Shelf (COTS) products such as PeopleSoft, mainframe legacy applications built on top of a Customer Information Control System (CICS) and modern J2EE web applications built using Oracle Application Server Portal. In addition, many of the applications interact with third party Application Service Providers (ASP) and provide historical information to a Data Warehouse (DW). All of these must be integrated and coordinated with the new SOA approach.

The University historically has integrated business systems using IBM MQ and FTP. This traditional approach resulted in a number of point-to-point (P2P) integrations. These P2P integrations are costly to maintain and have resulted in tight coupling between the consumer and provider. However, the existing environment was already using a light-weight message exchange state transfer (MEST) approach that is leveraged to allow for further innovation. An enterprise service bus (ESB)-a sub-type of a message bus architecture-provides for a more decoupled environment and, while it has a higher up-front cost of deployment, it was determined that the value of an ESB increases exponentially over time while the costs of extending the system remain linear and predictable [1].

The University's IT department has a small group of dedicated architects and software engineers, many of whom have developed expertise in the higher education domain through their tenure. Because the group is small, each member often takes on multiple roles including support and administration. Due to this, the IT department required a solution that would accomplish the following:

- Meet the ever increasing consumer demand by taking advantage of reusable services and composite applications
- Reduce or eliminate the P2P integrations
- Leverage existing assets and skills for improved operational efficiency

Solution

1. Overview

Service Oriented Architectures can be implemented using a number of different patterns and technologies. The conventional approach uses patterns outlined in the WS-* specifications and one can select from a broad spectrum of technology products ranging from open source solutions such as Apache ServiceMix to commercial suites such as Cape Clear and Sonic Software. Unfortunately, the WS-* specifications are in a constant state of flux and can overwhelm developers attempting to digest over 1300 pages of detail. For example, if adhering to the all specifications, it would take the following steps/tasks to implement a SOAP service:

1. Model the process using Business Process Modeling Notation (BPMN).
2. Define a service interface using WSDL and register the service with a Universal Description, Discovery and Integration (UDDI) repository.
3. Generate Business Process Execution Language (BPEL) scripts using the BPMN that access the services from the service registry.
4. Define policies governing access to the service using WS-Policy.

The commercial ESB suites in the market were evaluated and since the IT group is relatively small, the decision was made to look for a solution that would result in a low-friction system that could foster the type of innovation a small group of IT resources could undertake and wouldn't force the group into a centralized service ownership model that relied on a single vendor. The University's domain is extremely fluid, with process changes, application changes and integration changes; therefore what was needed was an overall architecture and strategy that was a reflection of the true nature of the university.

Since message requests already were established as the central transfer mechanism across the university, a RESTful or Resource-Oriented approach was employed to implement an SOA. REST is based on a small set of widely-accepted standards, such as HTTP and XML and requires far fewer development steps, toolkits and execution engines. The three key benefits of a RESTful approach to SOA include a lower cost of entry, quicker time to market, and flexible architecture. A Resource-Oriented approach goes beyond a RESTful approach and provides a deeper more extensible and transport independent foundation. While REST design patterns advocate the use of HTTP a Resource-Oriented architecture supports services connected to HTTP as well as transports such as JMS or SMTP.

Although several ESB implementations such as Codehaus Mule support REST, only 1060 NetKernel is built upon a Resource-Oriented Computing Platform [2] (hence, "ROC"). The core of resource-oriented computing is the separation of logical requests for information (resources) from the physical mechanism (code) which delivers the requests. Services built using ROC have proven to be small, simple, flexible and require less code compared conventional approaches. This made it the ideal technology to build a technology platform.

2. Technology Platform

NetKernel is resource-oriented middleware that provides core Enterprise Service Bus (ESB) capabilities, addressing, routing and data transformations and can act as a service registry orchestration engine. NetKernel is a rich offering and also provides advanced capabilities for XML transformations, cache management and multi-threaded processing along with multiple transport protocols, including HTTP and JMS and a SOAP engine enabling it to provision conventional web services. NetKernel is a solid foundation for heterogeneous enterprise integration.

Conceptually, NetKernel provides access to resources identified by a Universal Resource Identifier (URI) address. All URI addresses are managed within a logical address space. A REST-based micro-kernel handles all requests for resources, resolving the URI from the address space and returning a representation of that resource. Requests to the micro-kernel can also be made to create new resources or update or delete existing resources.

Physically, a NetKernel system is composed of modules which expose public service interfaces and resources via URI addresses. Similar to a Java EAR, a module contains source code and resource configurations. Modules may logically import another module's publicly exposed services and resources incorporating them into the address space. Since imports refer to a module's logical name and can specify a version number, multiple versions simultaneously can be run and updated in a live system. Service requests are injected into NetKernel by transports, which are event detectors residing at the edge of a system. Service consumers can send requests via any supported transport such as HTTP or JMS.

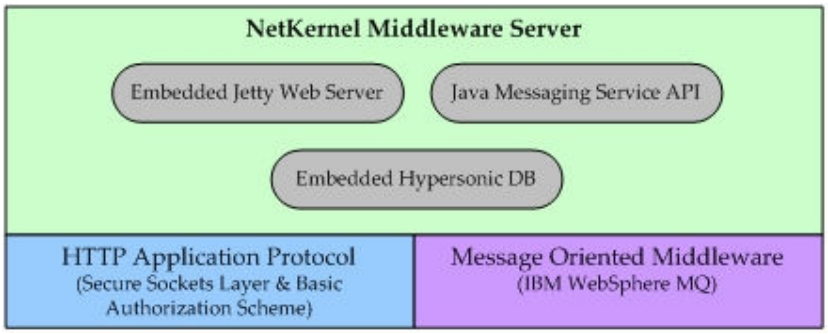


Figure 2 Technology Platform

HTTP is a stateless request-response application protocol. The *request* message structure is comprised of a command, headers, and a body. The *response* message is comprised of a status, header and a body. Clients and servers exchange these messages using Transmission Control Protocol (TCP) sockets. The client-server interactions can be secured at the transport layer using the Secure Sockets Layer (SSL) protocol while the message itself can be secured using encryption and a digital signature. Finally, a client can be authenticated using HTTP-Basic or HTTP-Digest authentication schemas [3].

The Java Message Service (JMS) API provides the platform for implementing asynchronous services. However, the API requires a provider, in this case IBM WebSphere MQ. IBM MQ is Message Oriented Middleware (MOM) and provides

queue-based communication channels among applications. Channels are implemented using Point-to-Point or Hub & Spoke topology. In addition to transporting messages, MQ can also handle workflow, process automation, data transformation, monitoring and management.

3. Resource-Oriented Services

A resource-oriented service provides transport-independent stateless access to a resource. A resource is an abstraction of information. For example, a student's registration is an abstraction which can have representational forms such as a web page, an XML document, or a PDF file. A service exposes each resource representation using a resource identifier or address. The resource identifier is actually a relative Universal Resource Indicator (URI). A URI is composed of two parts, the scheme (e.g. HTTP and FTP) and the address. The second part of the URI is relative. The address, /domain/account/45322, is relative until it is associated to a scheme such as http, http://domain/account45322.

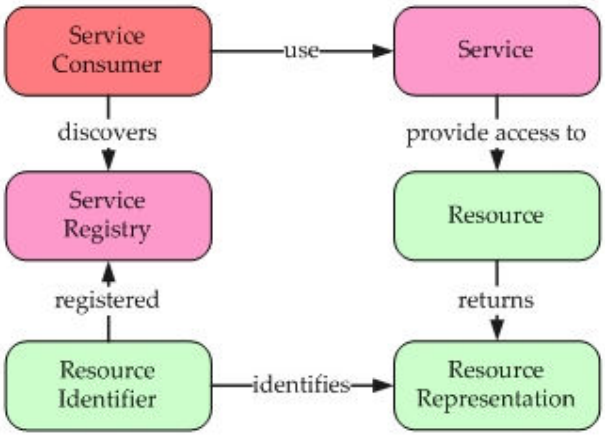


Figure 3 Resource Oriented Service Conceptual Model

The service defines a set of actions, *Create, Read, Update*, and *Delete*, that can be invoked on a resource. In addition, certain actions may trigger rules or business logic. When a resource is requested, a physical immutable representation of that abstract resource is returned. Different types of representations can be returned based on business logic, such as the type of consumer. For instance, most back-end processes require an XML document while a front-end process may require a JSON object. The service also can receive a representation creating a new resource, or updating an existing resource. Finally, the service can receive a request to delete a resource.

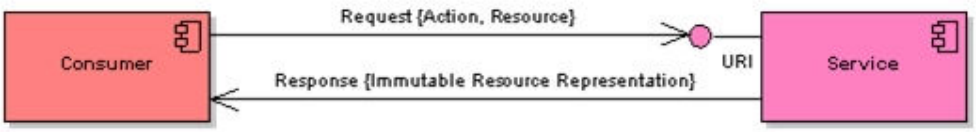


Figure 4 Resource Oriented Service Interaction Model

A transport resides at the edge of a system and when it detects an event it issues a corresponding internal root request. For example, the HTTP transport listens for a consumer's request for a Uniform Resource Locator (URL), specifying an access method (e.g. GET, PUT, POST, & DELETE). The URL is transformed into an internal relative URI and the access method is translated into an action. In the case of a request over JMS, the URI and action are passed as message header parameters. In addition, if a resource representation should be returned, a return queue parameter is specified in the header.

In a RESTful system, access to resources is stateless which means each request must have a way to pass context as meta-information. In general, the transport defines a message structure comprising a header and a body. The header is used to pass the meta-information while the body is used to pass the resource representation. For instance, if a

resource-oriented service is exposed over HTTP, authentication information can be passed in the header, and if the service is exposed over JMS that same information can be passed as an encrypted header parameter.

Resource-oriented services are built using Maven [4] and packaged as NetKernel Modules. Maven is a tool for building and managing software within the context of a "project." Maven projects are defined by a Project Object Model (POM) XML. The POM defines the software's dependencies, build process and deployment structure (e.g. JAR, WAR, EAR). In addition, a Maven project can be used to generate a project web site and deploy software. In this case, Maven packages services within a NetKernel Module and publishes information about those services to a registry.

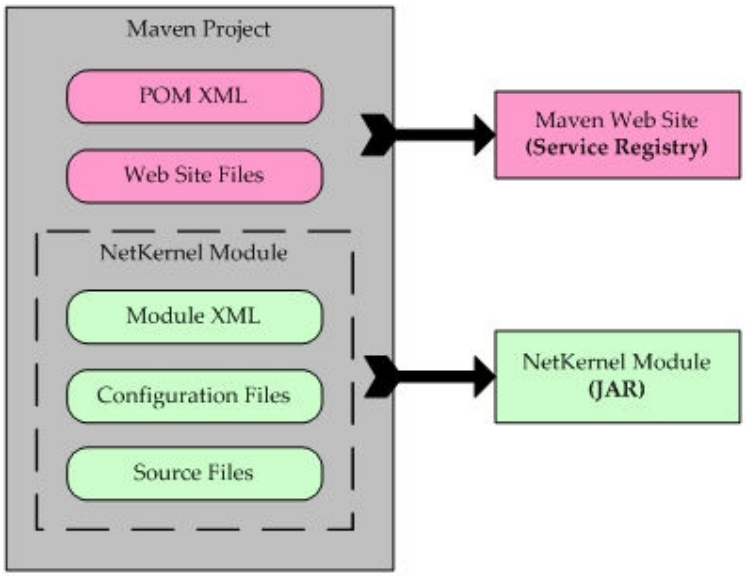


Figure 5 Resource Oriented Service Development

4. Resource-Oriented Enterprise Service Bus (ESB)

A Resource-Oriented Enterprise Service Bus (ESB) was implemented using NetKernel. At the heart of NetKernel is a RESTful or resource-oriented microkernel responsible for resolving logical URI requests to physical code endpoints and scheduling the request on an available CPU core. While the mapping of logical address to physical code is defined in the application structure, the actual binding of logical address to physical code occurs only for the duration of a request and is then discarded.

Because each request to the micro-kernel is bound anew, system administrators are free to change the association between logical address and physical code while the system is running enabling capabilities such as real-time live code updates. Performance in practice is paradoxically not degraded by this indirection and binding but rather enhanced as the URI address acts as a key to NetKernel internal cache. If any resource is re-requested and dependencies have not changed then the cached representation is returned instead of being recomputed.

The use of a resource-oriented microkernel has several key benefits. First, service interaction is at a logical rather than physical level. This results in loosely coupled interactions, thereby decreasing the impact to the consumer and provider when changes are made to the physical implementations. Second, the results of requests are cached, thus decreasing the overall cost of service composition and orchestration. For instance, if a set of orchestrated services relies on a common service, the underlying physical code of that common service seldom will be executed. Finally, all internal requests to the microkernel are asynchronous, allowing processing to scale linearly as more CPUs are added to the host server.

The ESB is primarily responsible for service provisioning and security. Service provisioning involves exposing resource-oriented services to consumers over transports such as HTTP and JMS. The transports map the external URI and access method to an internal resource-oriented service and action. Regardless of the transport, the body of the request, such as an XML Document or JSON object, is passed as a parameter named 'param.' Consequently, the resource-oriented services are decoupled from details of transport specific logic and in fact, additional protocols can be added at any time

without impacting existing code.

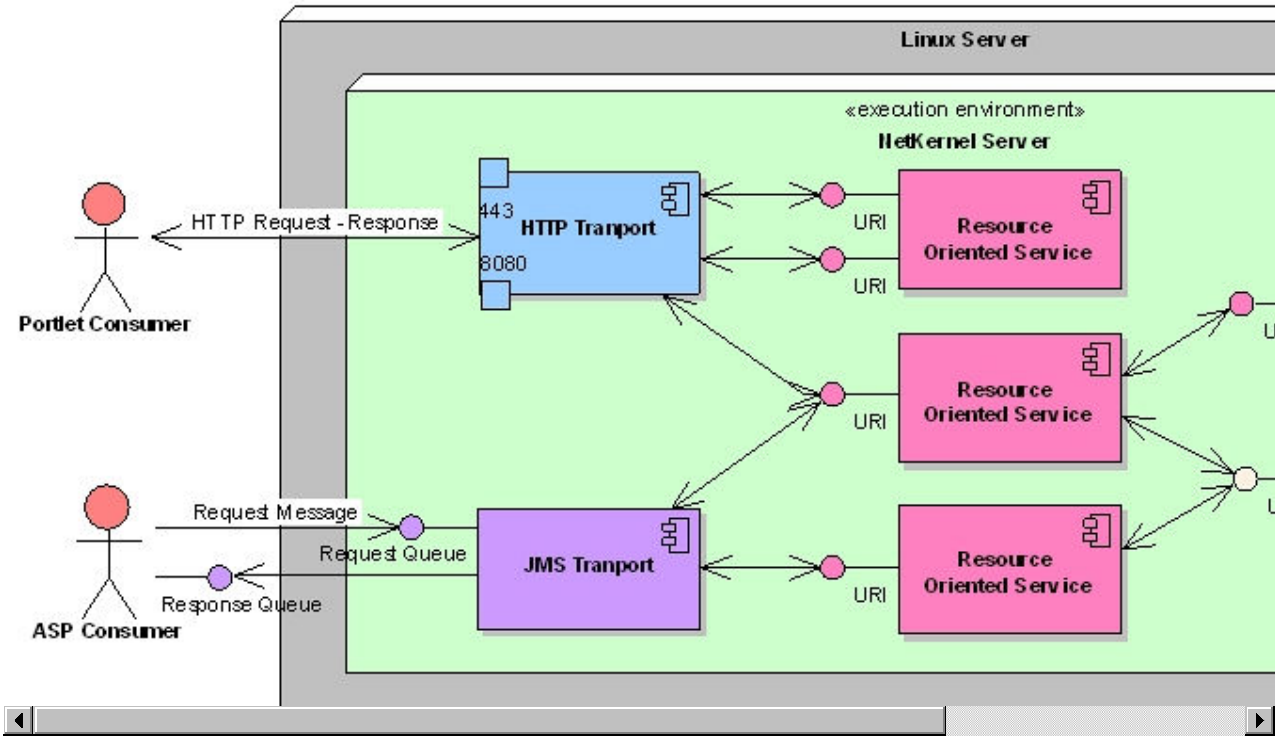


Figure 6 Resource Oriented Service Provisioning

The resource-oriented services, in turn, delegate to a set of custom infrastructure services and core services provided by NetKernel. NetKernel provides an extensive set of core services. For instance, core services exist for XML and SOAP processing, CRON Job scheduling, and SMTP interactions. The core services dramatically reduce the amount of code that needs to be written to implement a resource-oriented service. Custom infrastructure services are used to expose capabilities which can be leveraged in the higher education domain.

Each request to the ESB is first authenticated, then authorized and (in some cases) audited. Transports authenticate an incoming request based on a *user-name* password combination and then delegate authorization and auditing to a security service. Authorization involves verifying that the identified consumer has the appropriate privilege. A privilege is comprised of a relative URI and an action. As an example, a consumer could be authorized to read but not to update or delete the resource student profile, identified by the relative URI /domain/student/identifier/profile. Unauthorized requests automatically are audited and there exists the option to audit based on the identified consumer or privilege. The account, privilege and audit information is stored in an embedded Hypersonic database.

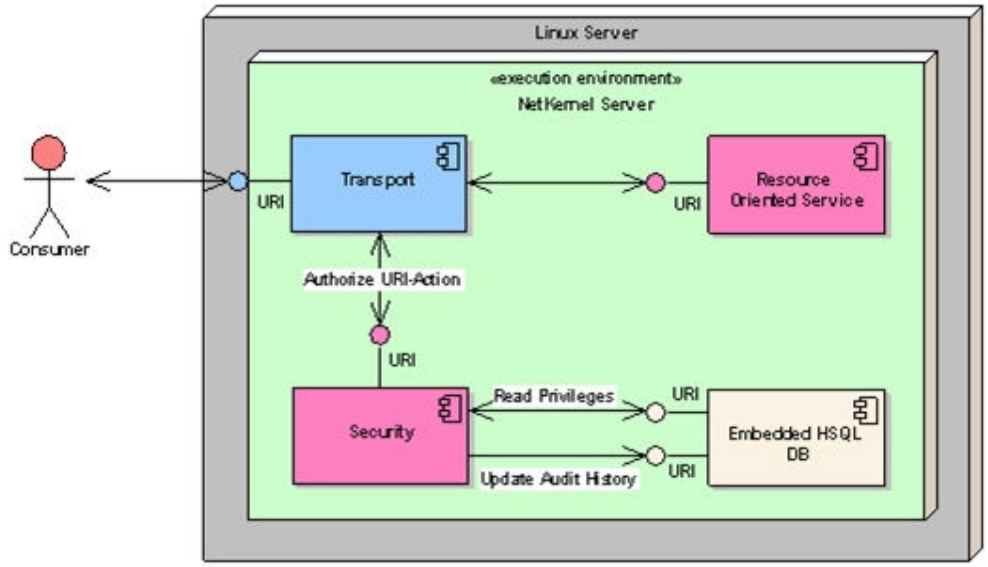


Figure 7 Resource Oriented Service Security

Summary

Other than a run-time service governance solution, such as Actional or ManagedMethods, middleware for implementing an ESB is essential to any successful SOA initiative. Without an ESB, an organization is simply increasing the number of costly point-to-point (P2P) interactions using web services. While there is wide disagreement as to the composition and purpose of an ESB, most can agree on a set of core capabilities including service addressing, message transformations and message routing. An ESB implemented using NetKernel middleware provides these capabilities and more advanced capabilities such as service registration and orchestration.

The NetKernel product allowed the University to implement a resource-oriented ESB. A resource-oriented ESB is essentially an open standards-based enterprise integration framework. This framework enables an enterprise to reduce or eliminate costly point-to-point interactions and decreases the time to market for introducing new capabilities. Furthermore, this framework has a lower initial cost of entry than a conventional enterprise integration framework based on the WS-* standards. Additionally, since NetKernel and ROC provide for integration on a per-service basis, the University can push the integration functionality to the edge of the network (as a URI), which translates into better service management and scalability. In short, this framework provides an organization unprecedented enterprise architectural agility.

In less than six months, a team of three software architects was able to implement a resource-oriented ESB and several initial resource-oriented services using NetKernel middleware. The successful implementation of the ESB launched the University's incremental adoption of SOA. A resource-oriented approach allowed the team to leveraged existing assets and technical skills. Going forward, the University's IT department now has the ability to meet the ever increasing consumer demand by taking advantage of reusable services and composite applications while reducing or eliminating the P2P integrations.

About the author

Jeremy Deane is a Technical Architect at Collaborative Consulting. He has over 12 years of software engineering experience in leadership positions. His areas of expertise include Enterprise Architecture, Performance Engineering and Software Process Improvement. In addition, he has Masters in Information Systems from Drexel University and has recently published a white paper, Document Centric SOA.

---

[1] Bottom Line SOA, The Economics of Agility by Marc Rix

[2] Introduction to Resource Oriented Computing
[3] HTTP Authentication: Basic and Digest Access Authentication
[4] Apache Maven Project

No comments Reply