Patterns for REST Services with WebSphere DataPower SOA Appliances

Daniel Colonnese
codaniel@us.ibm.com
WebSphere Technical Specialist
IBM, Cambridge MA

Alex Klevitsky
aklevitsky@mib.com
Director of Architecture and Enterprise Software
MIB, Inc. Westwood, MA

Abstract:

This article discusses service patterns using IBM WebSphere DataPower® SOA Appliances for Representational State Transfer (REST) style software systems. A reader is expected to have a familiarity with service-oriented software design and software design patterns fundamentals.

Patterns for REST Services with DataPower Appliances

This is a discussion of the role that IBM's WebSphere DataPower SOA appliances can play as an infrastructure component to implement SOA solutions based on the Representational State Transfer (REST) architecture. This is not a position paper about IBM's view of Service-Oriented Architecture (SOA) or the Enterprise Service Bus (ESB) portfolio. This vision is complimentary, rather than contradictory, to IBM's family of ESB solutions which includes WebSphere ESB, WebSphere Message Broker and WebSphere DataPower Integration Appliance XI50. While simplicity is the key feature of the proposed solution we would like to stress that business needs, rather than technology preferences, drive architectural decisions.

SOA Patterns and REST

"SOA is **flexible** architecture of **business capabilities** being supported by **loosely coupled** IT elements" as defined by the IBM SOA Center of Excellence. It is our opinion that this and other definitions present SOA as a set of architecture guiding principles, rather than reference architecture. Many definitions of SOA exist today attempting to formulate guiding principles for this new architecture. These principles alone are not new, but have been exposed as one cohesive unit. These guiding principles provide a foundation for the SOA reference model.

The *Exposed Router* variation of the *Exposed Broker* application pattern [1] provides the foundation for the SOA environment. According to L. Bass [2] a software system reflects not only requirements (both in terms of functionality and system qualities) but also the background and knowledge of the architects. Architects select the most common and familiar style that satisfies system requirements. The dominant style in large software systems for the past 30 years has been call-and-return architectures. The client-server is the most popular among them where programs are components and calls or RPC are connectors [2]. The client-dispatcher-server style "introduces an intermediate layer between clients and servers, the dispatcher component. It provides location transparency by means of a name service, and hides the details of the establishment of the communication connection between clients and servers." [3]   The client-dispatcher-server pattern describes a direct variant of the *Broker* pattern, indicating that servers and clients use and understand the same protocol. This paper further outlines exposing the DataPower appliance as a Dispatcher variant.

The benefits of SOA can be garnered by heterogeneous architectural styles which are created from SOA architectural patterns and REST [4].   REST is the predominant architectural style of distributed hypermedia systems including the modern Web. The DataPower SOA appliance expedites the implementation of software systems based on this style. The major driving factor in the rising interest of REST is attributed to the success of the World Wide Web – as of today the largest software system built by humanity. Roy Fielding's seminal dissertation illustrates REST data elements, components, connectors, and presents the following process view (Figure 1) in order to show the interaction between components [4].

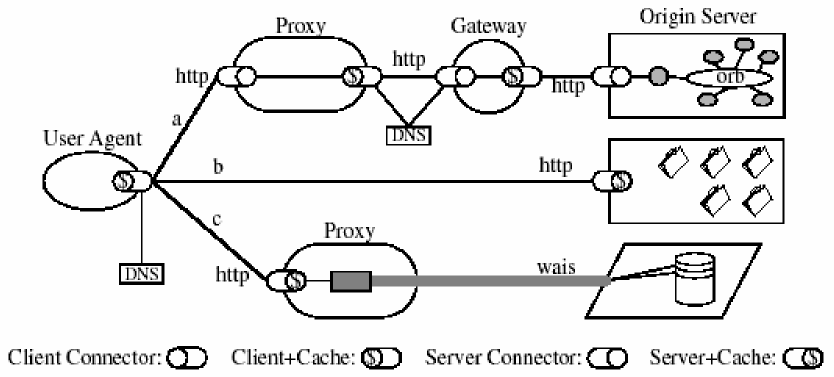

Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

Figure 1. Courtesy Roy Fielding PhD Dissertation, 2000

In his description of this process view, Fielding points out that "REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components and **intermediary components to reduce interaction latency, enforce security, and encapsulate legacy system**" (highlights in this quote are ours). Later he adds - "layered system constraints allow intermediaries – proxies, gateways and firewalls – to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching". In particular, "a *gateway* (a.k.a., *reverse proxy*) component is an intermediary imposed by the network or origin server to provide an interface encapsulation of other services, for data translation, security enforcement and acceleration. Note that the difference between a proxy and a gateway is that a client determines when it will use a proxy" [4]. The DataPower SOA appliance provides Multi-Protocol Gateway (MPGW) application component. This component is a good fit as REST gateway component implementations. It also represents variants of the “server-side proxies” – the participating components of the *Exposed Broker* pattern [3].

The recent popularity of SOA and REST fits the use of DataPower appliances in a Dispatcher or Gateway pattern thus leading to innovative software system designs and implementations.

DataPower- Gateway Design Patterns

This article is not concerned with specific implementation details; rather it is a position paper dealing with design alternatives. In a recently published book “RESTful Web Services” [5] the authors address design and implementation of Web Services based on the REST architectural style, taking into account functional requirements (including, but not limited to, security enforcement, protocol translation, data transformation and routing) as well as non-functional requirements (including, but not limited to, performance and consumability).

The introduction of the IBM DataPower appliance as a gateway offers flexibility in design choices which may otherwise be prohibitive or more complex in a software-only design. We offer three hierarchical variants of the gateway pattern based on the capabilities of the appliance: *Secure Gateway*, *Secure-Accelerator Gateway* and *Secure-Decorator Gateway* (Figure 2). Each lower level pattern helps to complete higher level pattern(s).

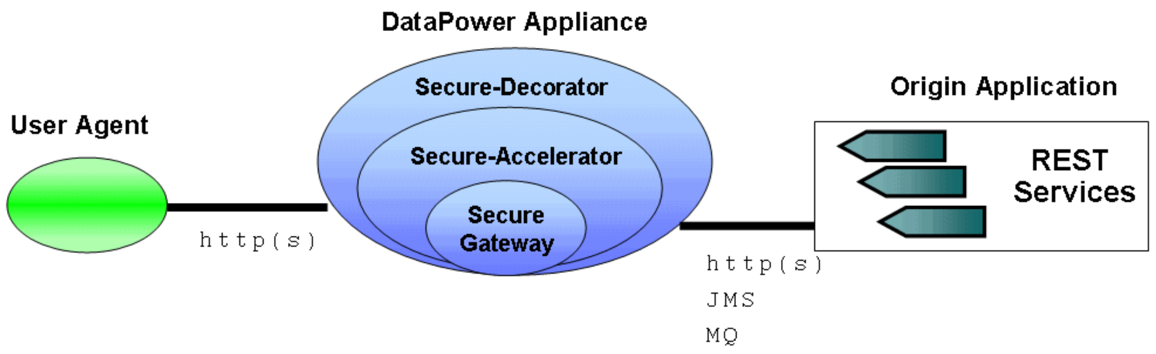

Figure 2. DataPower appliance as REST-Service Gateway

The *Secure Gateway* pattern utilizes DataPower to provide security services and protocol mediation. The DataPower Multi-Protocol Gateway Service (MPGW) utilizes DataPower security capabilities, such as AAA, CRL verification, traffic shaping, as well as XML threat protection. DataPower's integration capabilities include support for various messaging protocols (e.g. MQ, JMS) and FTP transports in addition to front-side HTTP(s). Multi-transport support is not violating REST according to the REST process view. The stateless nature of REST requires that a *Secure Gateway* assumes that all client side traffic is suspect and all back-end messages are authorized and valid.

The *Secure- Accelerator Gateway* completed by the *Secure Gateway* pattern adds data transformations and validation of XML messages. DataPower XML acceleration capabilities substantially increase performance of XML processing and provide highly efficient solutions for transforming programmable representations (e.g. XML) accepted from the client to the common internal XML message format (or business local XML vocabulary). We recommend including XML Schema compliance verification that is more expensive to perform utilizing other infrastructure components. We have found ISO Schematron [6] to be useful in implementing constraints and complex business rules validations in addition to basic XSD schema validation. With REST-AJAX web applications DataPower could be utilized to perform client-side validations via ISO Schematron or XPATH which can further reduce latency while decreasing load on the application server. The content-based routing capabilities of the appliance may be used to support versioning of the common internal XML message format.

The *Secure-Decorator Gateway* is completed by the *Secure- Accelerator Gateway* pattern which adds representation construction for human-readable (e.g. XHTML) and programmable (e.g. XML) interfaces from the common internal XML message format. It is a variant of a *Proxy-Decorator* pattern that was envisioned by the authors of "Design Patterns" as a combination of the well known *Proxy* and *Decorator* patterns [7].

Web 2.0 and beyond

Consumability (or connectedness) has been an important focus of Web 2.0 mashups. Many architects have stressed that REST services offer this consumability quality.

DataPower WS-Proxy and MPGW application components provide features to amplify this quality even further by potentially providing alternative support for non-REST Web Services. These patterns may be extended to add SOAP wrapper, WSDL, and WS-* capabilities to existing REST services.

Other new heterogeneous architectural styles may be built upon REST and selected SOA architectural patterns. A good resource is the IBM SOA Patterns Redbook [1].

In conclusion we would like to recommend that pattern based architectural styles and in fact software design patterns in general be used in the way Christopher Alexander advocated pattern usage in the field of civil architecture – to build better systems rather than simply solve recurring problems in a given context.

Resources

1. Patterns: SOA with an Enterprise Service Bus. IBM Redbook. May, 2005.
2. Len Bass, et al. Software Architecture in Practice. Software Engineering Institute (SEI). Addison-Wesley, 1998
3. Frank Buschmann, et al. A System of Patterns. John Wiley & Sons, 1996
4. Roy Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD Dissertation, 2000
5. Leonard Richardson, et al. RESTful Web Services. O'Reilly, 2007.
6. ISO Schematron: A language for making assertions about patterns found in XML documents. http://www.schematron.com/spec.html
7. Erich Gamma, et al. Design Patterns. Addison-Wesley, 1995.

About the authors

Daniel Colonnese is a WebSphere specialist specializing in helping Insurance and Financial Services clients design and build service-based applications.

Alex Klevitsky is the Director of Architecture and Enterprise Software at MIB, Inc. He has over twenty years of experience in design, development and implementation of Business and Engineering applications as well as Software tools.