Fiorano

Enabling change at the speed of thought

www.fiorano.com

REST-Based SOA

Efficient, Dynamic, Flexible SOA Implementations

AMERICA'S
Fiorano Software, Inc.
230 S. California Avenue,
Suite 103, Palo Alto,
CA 94306 USA
Tel: +1 650 326 1136
Fax: +1 646 607 5875
Toll-Free: +1 800 663 3621
Email: info@fiorano.com

EMEA
Fiorano Software Ltd.
3000 Hillswood Drive Hillswood
Business Park Chertsey Surrey
KT16 0RS UK
Tel: +44 (0) 1932 895005
Fax: +44 (0) 1932 325413
Email: info_uk@fiorano.com

APAC
Fiorano Software Pte. Ltd.
Level 42, Suntec Tower Three 8
Temasek Boulevard 038988
Singapore
Tel: +65 68292234
Fax: +65 68292235
Email: info_asiapac@fiorano.com

REST-Based SOA

Efficient, Dynamic, Flexible SOA Implementations

Over the past ten years, a vast number of companies worldwide have implemented Service Oriented Architecture (SOA). Just to recap, SOA is all about the deployment of business processes with the aid of pre-built, pre-tested 'business services', the idea being that visual software tools can be used to wire-together Services on a screen to implement particular business functionality.

SOA promised faster deployment cycles, enhanced flexibility and responsiveness, easier change management and alignment of business and IT functions. The ideal was that whenever a business requirement changes, an SOA-enabled organization would be able to implement the change rapidly by simply 'rewiring' pre-tested business services on a screen, reducing the normal costs of programming and deployment of a solution using traditional, non-SOA methods.

Unfortunately, experiences in the past decade have shown that the vast majority of SOA projects have run well over budget and that many have failed outright. In the cases that have been successful, costs of both software licenses and professional services (to implement the SOA) have been inordinately high, limiting the successes to large enterprises able to commit serious levels of resource and time to achieve their goals. Successful SOA implementations have been expensive for many reasons, with some of the critical ones being:

- **Inherently Complex Technology**: The vast majority of "SOA Stacks" used in SOA implementations today are based on older BPM-style approaches, with a central hub maintaining the state of business processes while end-point services implement process steps. Implementing integration processes with such an approach requires inordinate amounts of programming, greatly increasing professional-services outlays even for simple projects. On average, the consulting outlay for a successful SOA project has been between 4x to 7x of license cost, signifcantly eroding target ROI on such projects.

- **Change Management Issues**: Constraints of the underlying technology also make it difficult to implement and manage business process change. Since the tools are inflexible (because of inherent flaws in the underlying implementation technology), changes are difficult to implement and maintain. In the typical case, significant programming/consulting time is required for each implementation change, with multiple develop/test/deploy cycles. In addition, projects are disrupted because the core software does not allow dynamic changes to processes.

- **Scaling Problems**: With current approaches, scaling implementations (as more users are added or more processes deployed), is very expensive. For the most part, all existing SOA technology is hub/spoke based and does not scale linearly. When the limits of a given hardware machine are reached, an additional 'hub' must be added, incurring massive software costs. Scaling also requires disruptions since in the typical case the entire system must at some point be stopped, changed/modified and then redeployed across and additional number of hubs: not an easy process. Existing software stacks based on BPM-style approaches make scaling particularly difficult since there is no easy or seamless way to deploy a portion of a business process on a new hub, should the need arise.

- **Architecting for the Cloud**: In addition, with the proliferation of cloud-based technologies, enterprises attempting to implement a traditional stack will find further elements of rigidity in the system. Traditional stacks were not built with the distributed, ubiquitous enterprise in mind. In order to achieve the elasticity benefit of the Cloud for a distributed application, it is essential for the application tier to be stateless. It is an important best practice that all application logic in the Cloud should be stateless. No object instances, no session Beans, no server cookies. If the load on your application spikes, the Cloud should respond elastically by provisioning adequate resources to meet the need. The more stateless your application is, the better able the Cloud will be to achieve this seamless elasticity.

  The Cloud may need to spawn additional instances to handle the load, and any particular instance may crash. But because the Cloud is highly available and partition tolerant, such a crash must not disturb the process that Cloud instance is supporting.

REST (Representational state transfer)-based SOA is an emerging technique of implementing SOA projects that resolves the three critical issues discussed above. Rest-based SOA is centered around document-centric, event-driven, loosely coupled, asynchronous, message-based Business Services.

By combining a simplified service-model with event-driven messaging, Rest-based SOA allows the deployment of linearly scalable, easy-to-change, manageable SOA projects at a fraction of the cost associated with traditional SOA-stack implementations.

REST-based SOA – Basic Principles

While document style interfaces are the norm for generic, Web Services-centric SOA implementations, REST-based SOA takes the notion to a whole new level. In the traditional web-services centric approach, document-style interfaces are described in a WSDL file, with the constraints on input and output messages being described by strongly-typed XML schemas. The consequent tight-coupling mandates an RPC-style of programming with centralized state management, impeding scalability and flexibility.

The REST-based SOA approach relies on three fundamental principles: messages, documents and content based routing.

All communication between Service end-points occurs via ***messages***. Distributed message-processing is a critical component of a REST-based SOA system. Inside messages are ***documents***; the REST-based approach does not use Remote Procedure call or anything remotely like it. RPC is popular because it is easy for programmers to understand and view a problem space using this method. When RPC goes away and you have documents and messages it's a different way to think about and view the problem – one has to solve the problem with a combination of asynchronous communication and content-based routing. An important characteristic of documents is that they must be ***human-readable***: If you stop all the computers and hand out the documents to humans they should be meaningful.

Content based routing is the third fundamental principle of REST-based SOA architecture: You don't know the end point of your messages by network addresses, or server names. You don't address individual resources by network addresses; rather, as described in subsequent sections of this paper, you insert URI's in the document and that gets it to its destination via late-binding at runtime.

Document-centric Interfaces and State Management

In a REST-based SOA architecture, Service-interfaces are not based on any formal contract, relying instead of the GET, PUT, POST and DELETE operations familiar from HTTP. Messages containing documents flow between Services, typically distributed via an asynchronous, message-based ESB infrastructure.

Documents as Interfaces

In REST-based SOA, the Document ***is*** the interface. There's nothing else to rely on for interfaces in this architecture – no protocols (such as WSDL) are required. The developer of a Service must express the interfaces in the document, which are understood by downstream Services via embedded message-formats. This mechanism removes the need for defining a specific, formal contract between Services, enabling a late-binding, loosely-typed, easier to modify and efficient implementation resulting in lower maintenance and development costs.

A Service Oriented Approach to State Management

Further, in the REST-based approach, all state information is carried within the document. As they flow across Services in a process, documents are dynamically augmented with state information and persisted locally, obviating the need for a centralized state repository. Information is never lost due to a system shutdown for any reason. On restart, processing picks up where it left off, making a REST-based SOA implementation significantly more scalable than traditional approaches.

Document characteristics and practices

Documents in a REST-based SOA system are self-describing. Documents are not dependent upon any particular stack of software to be used or to be useful. They are completely neutral to the Services that consume and produce them. XML is not an explicit requirement although in most large projects several documents tend to be XML-based.

Documents specify the contract between services. In REST-based SOA, this is not done explicitly; since documents are self-describing, Services that process a document implicitly understand the message-structures embedded within the document, obviating the requirement for explicit interfaces such as WSDL.

Since Documents describe the interfaces between Services, they are created first – before any Services - in any REST-based SOA system. A good rule-of-thumb in the REST-based approach is to be able to express all document characteristics in a non-XML syntax. This ensures that documents are simple structures, typically hand-crafted and easily understood, ideally avoiding references, "includes", namespaces and other XML-centric features.

The focus in a REST-based SOA implementation is to trade off build-time purity for on run-time simplicity, mandating an asynchronous, late-binding, message-centric approach to implementation.

Resources and Logical Routing

All sources of information in a REST-based SOA implementation – whether they are business processes, documents, service entry or exit points or anything else – are ***resources***. All resources are addressed by URI's which are very much like a Web-URL, conformant with W3C naming specifications. URI's are used across domains (such as, for example 'development', 'test', 'production', etc.). All URI's in the system are registered with the underlying ESB infrastructure, allowing documents to be routed and filtered and policies to be enforced via URI's, as described in more detail in the following section.

Figure 1 illustrates a simple Document structure with embedded URI's.

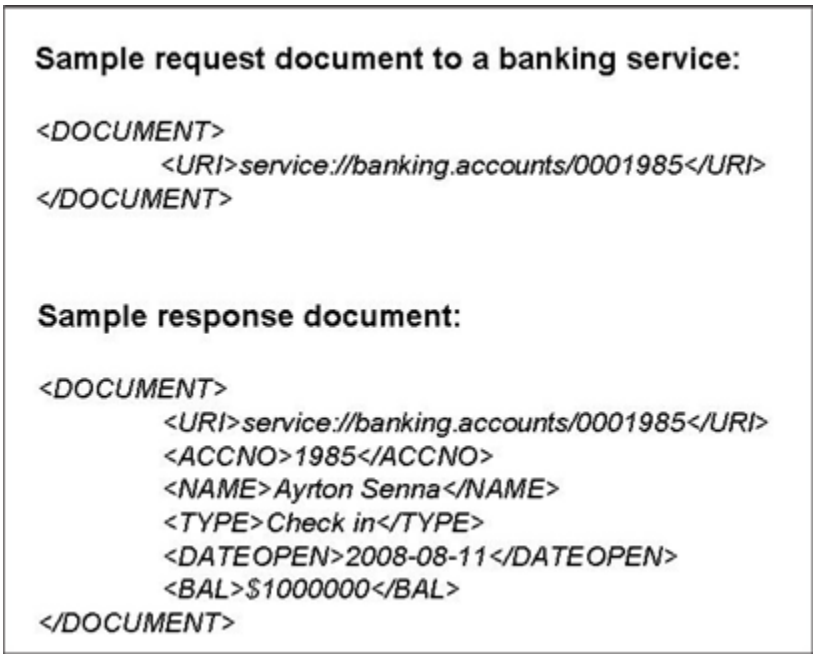

Figure 1: Document sample, with embedded URI's

Late Binding – dynamic, runtime extensibility

There is no build-time mapping to particular resources in a REST-based SOA system. Services sending and listening for documents can talk to any message-types and senders can send documents anywhere routable. At runtime, the ESB infrastructure routes documents based on content and URI's to the appropriate destination. In contrast, in the traditional approach most SOAP endpoints point to a WSDL which contains an IP address or server name with a specifically programmed interface, creating a fairly tight coupling that is fragile.

In a REST-based SOA system:

- Messages can be routed to any resource – there is no 1 to 1 mapping or tight coupling of any kind.
- Resources are loosely coupled to message contents - loose coupling is extremely important to SOA; So each resource can handle many types of messages.
- Reliance is placed on dynamic typing – in other words, service endpoints need to be able to handle lots of different message types. A particular service-endpoint may not be able to process each particular incoming message-type but it needs to be able to handle them in conjunction with a rich, underlying message-driven ESB infrastructure.

Strong typing and early binding vs. loose coupling and late binding

The REST-based approach described above does not work well with SOAP for obvious reasons. A SOAP call is statically bound to a particular version of WSDL, making it inherently inflexible.

The difference between traditional SOAP/WSDL centric SOA and REST-based SOA boils down to a classic argument between strong typing and early binding versus loose coupling and late binding from the programming languages world (C++ vs. Smalltalk). In the late-binding approach, you don't get errors at compile time and then have to do a lot of debugging when your program fails at runtime. This problem does not occur in an SOA implementation that is supported with a rich run-time environment with support for asynchronous messaging and powerful, dynamic exception handling. Service end-points can throw exceptions for unhandled message types, hand the message back to the ESB and nothing rolls over and dies. The resulting system is inherently more flexible and dynamically extensible.

Asynchronous Messaging: the foundation of REST-based SOA Platforms

Late binding and logical routing mandate an underlying infrastructure that supports asynchronous messaging at its core, preferably with a distributed architecture for scalability.

With an Asynchronous system, there are no blocking calls and no waiting for responses, making the system inherently more scalable than a blocking, synchronous system. ***Message-senders are 'fire and forget'***; they only publish the messages to the ESB and then move to the next step. ***Message-listeners are like handlers***: they wait for an event and when that event occurs, they know what to do with the embedded document in the event.

From a programming perspective, this approach is very different from the synchronous, RPC-style approach. However, most asynchronous systems do support pseudo-synchronous styles of programming, allowing some flexibility in this regard.

With an asynchronous approach, if a message needs to go through a series of steps in a process and the power dies or there's a processing problem, you can pick up where you left off since messages may be optionally persisted to disk at every step, providing ***assured delivery***; there is no central state repository, making the system inherently more scalable.

Summary of Business Benefits of using REST-based SOA

Technology must ultimately serve business interests to be viable in a commercial environment. REST-based SOA, implemented with good architecture provides the following key benefits to the business:

REST-based SOA	ROI factors for business
1. Easier to implement & maintain 2. Reduces complexities	• Significantly less reliance on and reduced cost on professional services • Faster time to market
3. Linearly scalable system	• Extend the system with build as you go functionality instead of a huge upfront investment in a monolithic system
4. No centralized bottleneck	• Reduced cost and risk of failure • Efficient use of resources through load-balancing across multiple distributed peer-servers • Reduced hardware costs
5. Flexible and dynamically extensible	• As decision-making data points change, change the system rapidly to incorporate new business intelligence – no huge IT intervention and no extended professional services required
6. Reuse services	• Significantly reduce costs through reuse of services
7. Dynamic exception handling	• Less downtime via easy extensibility.
8. Stateless application architecture suitable for cloud applications	• Provides a perfect platform for elastic, distributed cloud applications.

REST-based SOA and Fiorano

The Fiorano ESB and SOA platform provides built in support for all of the concepts discussed in this paper, including document-centric interfaces, a service-oriented approach to state-management, logical routing based on URI's, late binding and dynamic, extensible exception handling, together with a runtime environment with support for both queuing (point to point) and publish/subscribe (one to many) communication with associated productivity tools. For more information, please visit www.fiorano.com.