Technical white paper

An introduction to SOA and the HP NonStop server environment

Table of contents

About this document

This document provides introductory information about the basic business rationale, concepts, and principles of service-oriented architecture (SOA), as well as gives an overview of how SOA principles and technologies can be applied to and are supported for applications running in the HP Integrity NonStop server environment.

SOA is everywhere

SOA is the three-letter acronym making the rounds of the IT community for a few years now. The marketing literature for all major software vendors and analyst reports seem to evangelize the virtues of SOA, with visions of greater business agility and IT nirvana. Industry conferences and trade journals are replete with references to SOA, and industry consultants espouse the benefits of SOA. So, what exactly is SOA and why should you care about it?

What is SOA?

While SOA does indeed require technology and products at an infrastructure level, it is not an off-the-shelf product. SOA is not an off-the-shelf product but a concept.

At its core, SOA is about using a new approach—to build IT systems that promote design and use of loosely coupled modular components ("services" in SOA terms) to support the IT enablement of business processes. In a SOA environment, these services are available on a network and can be accessed using standardized mechanisms without knowledge of their underlying implementation, including the choice of technology that has been made to implement the service. This fact, coupled with certain design principles that are elaborated on later in this paper, enables and encourages service reuse.

Service reuse is a central benefit of the SOA approach because it enables you to leverage your current functional software assets and thus reduce the time needed to respond to business changes. This, in turn, improves business agility. While the concept of software reuse is not new and has been promised by earlier programming paradigms, in general the benefits have not been realized. The SOA approach, however, provides both implementation guidance (for creating small service modules) and operational standards (for platform independence, automatic discovery, and use of service modules). So with SOA, the promise of service reuse has a solid underlying foundation. Service reuse is a theme that we will return to often in this paper.

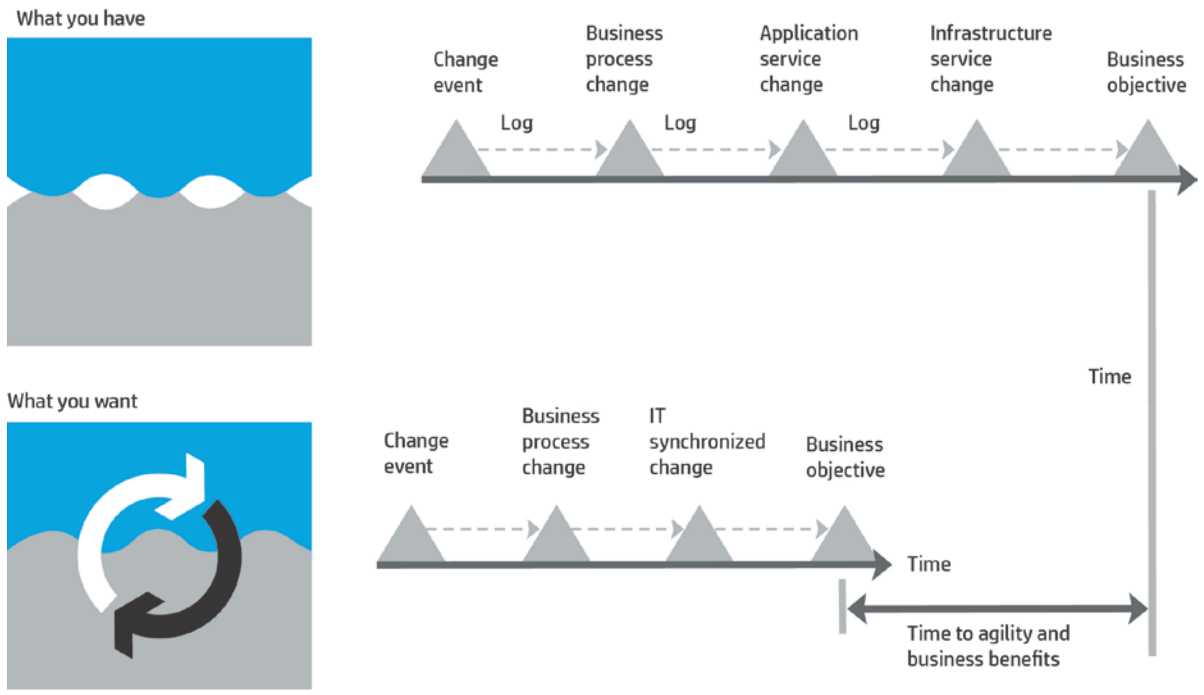
Why should you care about SOA?

Put simply, the fundamental reason you should consider a SOA approach is so that you can move your organization toward an agile enterprise. The approach will help IT respond to constantly changing business needs more quickly and can do so with fewer resources. In other words, SOA enables IT to reach a state in which business and IT are synchronized to capitalize on change, which translates to increased business agility. SOA achieves these goals by providing a methodology for application development that:

- Enables existing applications to work better together
- Preserves existing investments
- Allows new business processes to be deployed quickly

The goal of SOA is shown in figure 1.

Figure 1. The goal of SOA—business agility



What is a service?

Because the notion of services is such a central concept in SOA, it is helpful to gain a basic understanding of the term service. In the context of SOA, a "service" constitutes the basic unit of business automation logic and is defined by means of a standard service description document. This standard service description document is defined using Web Services Description Language (WSDL) and forms a "service contract" between the service provider and the service consumer. The service contract defines the operations offered by the service and includes rules that must be met and accepted by a service consumer in order to establish successful communication and interaction with the service provider.

When a service consumer requests an operation from the service provider, the service consumer does not know, nor necessarily care, how the service implements the requested business action. The consumer cares only that the service performs what is defined by its published service contract.

Note that the service contract shields the service consumer from knowing, or, worse, from being dependent upon, the implementation details behind a service. So, the key concept behind a service is that the service implementation is encapsulated behind a standard, published contract that describes the functionality that a service operation exposes to its consumers. This hiding of the implementation process provides a degree of freedom, in terms of how a service can be implemented; it also offers the potential to reuse existing software assets.

This concept of hiding an implementation behind a well-defined contract is not new. CORBA, with its Interface Definition Language (IDL) structure, and Java technology, with its interface class, achieve the same objective. But in the case of CORBA and Java, the transport protocol is technology dependent (Internet Inter-ORB Protocol [IIOP] in the case of CORBA, and Remote Method Invocation [RMI] in the case of Java), this forces technology awareness between the service provider and consumer. While the transport protocol in SOA is technology neutral, thus allowing operations from service providers and consumers to be implemented using any technology that is right for the job.

Service contracts, specified by WSDL, enable a service to provide a formal definition of:

- The service endpoint, such as an HTTP port or a Java Message Service (JMS) queue, which specifies the service access point in the network
- Service operations, which is the set of functions that the service wishes to expose and which are used by the service consumer to interact with the service
- Every input and output parameter supported by each operation that defines the syntax of the operation
- Rules and characteristics of the service and its operations, such as whether an operation can engage in a dialog with the service consumer or whether the interaction is strictly a request or response operation

In other words, services encapsulate the logic behind a service contract and provide the basic building blocks for automating business tasks.

Service characteristics

Now, how should services be designed in order to attain the potential benefits (such as business agility, service reusability) of SOA?

The important question being, what are the design principles, if any, that one must adhere to while designing a service? This question is important because merely implementing SOA does not necessarily lead to meeting the expected business benefits.

In our discussion about the principles of service construction, we are not concerned with the technology and platform requirements for implementing the service, because SOA does not dictate the implementation details behind a service. Service consumers do not care, nor do they need to know about those details.

While consultants and practitioners have varying opinions about the constituent principles behind service construction, there is general consensus on the following core principles:

- Services are loosely coupled
- Services abstract underlying logic

The principles behind service construction

Loose coupling

SOA is based on the concept that the solution logic is partitioned into services that can be assembled to automate business tasks collectively. So, what should be the dependency among these collaborating services?

This is where coupling enters the picture. “Coupling” is a measure of the interdependency between units of logic that, in the case of SOA, are services. It should be a design goal to reduce the dependency among services. This is achieved by limiting the dependency between a service provider and its consumers to the information expressed in the service contract. This also implies that the service contract should be designed in a way that is not specific to any one service consumer.

This principle of loose coupling is an important one for responding effectively to the inevitable changes in the IT environment that are necessary in dealing with changing business requirements. Loose coupling reduces the time needed to respond to business requirements by helping eliminate the ripple effect caused by such things as changes in the implementation logic of a service. By definition, loose coupling implies that the implementation of the changed service does not have a dependency on any of its collaborating services.

Service abstraction

Put simply, it is the ability of the service to inform the outside world about those service capabilities that it wishes to expose solely through a service contract, thereby hiding the implementation logic from the outside world.

By paying close attention to this principle and designing a well-crafted service contract, in a generic manner that can accommodate multiple service consumers, the service can be positioned as a reusable asset. Reuse fosters agility because it reduces the time needed to roll out new services, thereby reducing the time needed to respond to changes in business requirements.

It is also the reuse principle that enables services to be implemented as "black boxes," with the operations in the service contract providing the sole means of interacting with the service. Black boxing a service makes it easy to make changes to implementation logic because, as long as the changes do not alter the service contract, service consumers are not impacted by changes made to the implementation.

All this places a great deal of emphasis on how service contracts are designed. The greater the abstraction, the more generic the service contract can become, and the higher the service's reuse potential.

While loose coupling and service abstraction form the core design principles of SOA and receive the most attention in SOA literature, there are other design principles that are also important to fostering service reuse and thus producing increasing business agility. For SOA, services also should be:

- Amenable to composition
- Autonomous
- Stateless

Service composition

Service composition refers to the ability of a service to be part of another service. The need for service composition comes from two sources. First, as your experience with SOA projects increases, inevitably the number of service assets in an organization grows, and there is a need to define some sort of service hierarchy in which one service is composed of other services. Second, as the sophistication of your SOA projects increases, you may want to orchestrate the individual services using a service-oriented business process that is expressed through a composition language such as the Business Process Execution Language (BPEL). In this case, the high-level business process is a composition of individual services.

It is important to realize that the ability of a service to be composed is another form of reuse, and, therefore, its service operations must be designed with an appropriate level of granularity to increase composition opportunities. The requirement for any service to be able to participate in service composition places an emphasis on the design of service operations.

While the motivation for such design principles was driven by the principle of reuse, the motivation for the next two design principles—service autonomy and service statelessness—is a consequence of service reuse. Increasing reuse of a service means that, over time, the service becomes part of many service compositions that eventually translates into high usage volumes and unpredictable usage scenarios. While the runtime behavior of services under heavy load is determined to a large extent by the robustness of the underlying service container infrastructure—something at which the Integrity NonStop platform and its middleware excel—the service design principles enumerated below also play a significant part.

Service autonomy

Service autonomy is a measure of the degree of control that a service has over the resources on which it relies for performing its function. The greater the degree of control, the greater the service autonomy. By control, we mean whether the resources that the service must rely on are also shared by other services or other software entities within the enterprise.

By increasing the amount of control a service has over its own execution environment, you can reduce the dependencies the service may otherwise require on shared resources within the enterprise. The concept of allowing a service to control its own destiny enables the service to be designed so that it provides the right quality of service levels, both in terms of performance and scalability. Even though it is not always possible to provide a service with exclusive ownership of the resources it encapsulates, the primary design objective should be to provide the service with a reasonable level of control over the resources required for its execution.

There are two kinds of service autonomy that should be taken into consideration: service-level autonomy and pure autonomy.

Service-level autonomy

In this case, no two services share the same resource, but a service may still share underlying resources. For example, a wrapper service that encapsulates a Pathway server resource that is also used independently from the service, has service-level autonomy. The wrapper service shares the Pathway server resource with other Pathway clients.

Pure autonomy

With pure autonomy, the underlying resources are under complete control and ownership of the service. This is typically the case when the implementation is built from the ground up in support of the service.

Implementing a purely autonomous service helps in dealing with scalability concerns, such as concurrent access conditions, as well as making it possible to provide predictable service performance. However, this approach may require more hardware and software resources, resulting in larger, more complex systems. The appropriate level of service autonomy depends upon the service being considered, and is a trade-off between the particular service-level requirements and other considerations.

Statelessness

As the processing demands on services being reused, composed, and accessed concurrently increase on a regular basis, so does the need to enhance the service processing logic. When designing services that are expected to experience a high degree of simultaneous service consumer access, extra attention should be paid to state management.

The principle of statelessness indicates that services should be designed to both reduce the amount of state information they manage, as well as the duration for which the services remain stateful. In a service-oriented solution, state information usually represents data specific to a current service activity. While a service is processing a message, for example, it is temporarily stateful. If a service is responsible for remaining stateful for longer periods of time, its ability to remain available to other concurrent consumers is impeded.

As with service autonomy, statelessness is a preferred condition for services and one that promotes reusability and scalability. For a service to retain as little state as possible, its underlying service logic must be designed with stateless processing considerations.

How does SOA promote business agility?

We noted in the beginning of this paper, the reason that SOA is so popular is because the SOA approach encourages reuse, which speeds responsiveness to changing business requirements. So, how do the architectural principles help to meet the business goals of increased agility?

- Service loose coupling establishes an inherent independence that frees a service from immediate ties to other services. This makes it easier to achieve reuse.
- Service abstraction fosters reuse because it establishes the black box concept. Proprietary processing details are hidden and potential consumers are made aware only of an access point represented by a generic public interface.
- Service composition is primarily possible because of reuse. The ability of new service requirements to be fulfilled through the composition of existing services is feasible when the services being composed are designed for reuse.

- Service autonomy establishes an execution environment that supports reuse because the service achieves increased control over its execution environment, thereby fostering a predictable service performance.
- Service statelessness supports reuse because it enhances the availability of a service and typically promotes a generic service design that reduces state management and moves activity-specific processing outside the service boundary.
- IT applications designed to these principles enable a faster and less costly response to changing business needs.

Basic SOA building blocks

Having discussed the general rationale and principles behind SOA, and why you should care about it, this section will elaborate on the technologies underlying the SOA model.

The approach to the actual service implementation (business logic) does not change significantly if a SOA methodology is being employed. The usual attributes of scalability and availability are required for the service, and, to achieve these characteristics, the service should be deployed in a scalable and available infrastructure.

In the case of the Integrity NonStop server, this means implementing the service in one of the available application environments. Pathway, HP NonStop Servlets for JavaServer Pages software, HP NonStop CORBA software, HP NonStop Processes and HP NonStop CORBA software can all be used to implement SOA services, which may then be presented and used to compose business processes. Which of these application service models is chosen depends on many factors and is an implementation-specific decision.

A SOA application service may be an existing or a new application component. It can be implemented using any programming model, language, and execution environment that is appropriate for the application. For example, it could be a COBOL Pathway application, a C++ NonStop CORBA object, a Java application (Plain Old Java Object [POJO], Java Servlet, JSP, or Enterprise JavaBeans [EJB]), or a C NonStop Tuxedo service.

What makes an application component a SOA service is presenting an interface to the application defined by a well-known service abstraction. To enable this, a set of industry standards have evolved that provide the necessary underpinnings to deliver SOA services (for example, they enable the definition of the contract between the service provider and the service consumer). These underpinnings address the following core aspects of providing a SOA service:

- Description: how SOA service interfaces are described
- Messaging: how SOA services are invoked
- Discovery: how SOA services are found

Description: how SOA service interfaces are described

In a SOA architecture, no matter how the application is implemented, it must provide a service interface definition using the WSDL. WSDL is a widely adopted World Wide Web Consortium (W3C) industry standard that describes the public interface to the service. WSDL service definitions are expressed in XML format and define the protocol bindings and message formats that must be used to access the service.

One of the components of the WSDL service definition, for example, is a description of all the methods provided by the service, and addressing information needed to access the service. The WSDL interface definition is published (made public, or "exposed") in some way by the SOA service provider, and is then used by the SOA service consumer to construct the necessary messages to invoke the SOA service.

Messaging: how SOA services are invoked

The WSDL service definition also describes the messaging protocol or protocols that must be used to access the service. While, in theory, any protocol could be used to access a service, it makes sense to use some widely available standard protocol, which enables the service to be used by the widest possible range of service consumers and helps achieve the maximum benefit of the SOA approach (through component reuse, and more).

The standard method for SOA service access is the Simple Object Access Protocol (SOAP), which is carried over the Web standard HTTP transport. In most cases, the WSDL for the service describes the service access protocol in terms of SOAP over HTTP, although it is possible to carry SOAP message payloads over other transport protocols (for example, a reliable messaging transport). SOAP and HTTP are both also W3C standards.

Discovery: how SOA services are found

It is of no use to provide a SOA service if its existence is not made known in some way, so that it can be used by service consumers. There are really two basic ways to publish the SOA service WSDL description: either through some private means or through a public registry.

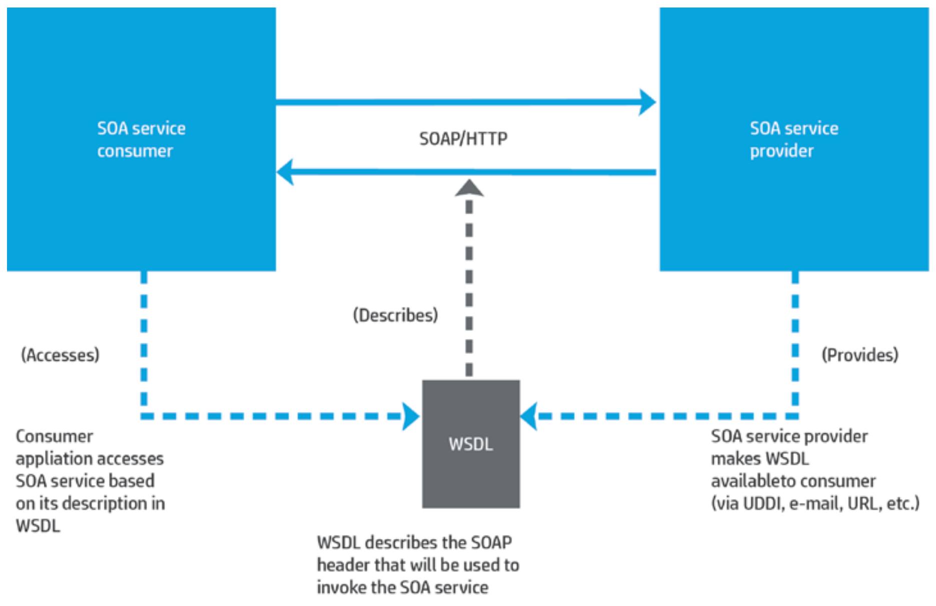
An example of the latter scenario is the Universal Description Discovery and Integration (UDDI) standard. SOA service providers can publish their interface definitions to a well-known UDDI registry, and SOA service consumers can discover and invoke services dynamically by querying the registry. In practice, it is more likely that the service provider makes the service definition available to the consumer directly, by some private means, at the time when the consumer is being developed, as opposed to the dynamic discovery and invocation method.

SOA services interoperability

Service reuse requires pervasive accessibility (interoperability), regardless of the hardware or software platform on which the service provider and the service consumer are executing. This goal is not achievable unless the required standards are agreed to and made readily available on a wide range of hardware and software platforms. To this end, the Web Services Interoperability (WS-I) organization was created by the major industry players, with the goal of helping to ensure interoperability between SOA services.

WS-I has developed Web services standard profiles. Support for a WS-I Web services standard profile by any platform facilitates interoperability with all other platforms supporting that profile. WS-I has been very successful in achieving its goals, and the standards comprising the WS-I Basic Profile have been adopted widely and are considered the core technologies supporting SOA, namely: WSDL, SOAP, HTTP, HTTPS, and UDDI.

Figure 2. Basic SOA building blocks


SOA service consumer
SOAP/HTTP
SOA service provider
(Describes)
(Accesses)
(Provides)
WSDL
Consumer appliation accesses SOA service based on its description in WSDL
SOA service provider makes WSDL availableto consumer (via UDDI, e-mail, URL, etc.)
WSDL describes the SOAP header that will be used to invoke the SOA service

As the SOA model is being used more widely, new Web Services standards are being developed and existing standards are being enhanced to provide additional features.

The basic building blocks of the SOA model are shown in figure 2.

Extended SOA infrastructure

The Web services standards described earlier provide the basic elements for exposing new or existing application objects as SOA services. Broader exploitation of the SOA model, however, necessitates the means for greater overall coordination and control of SOA business processes.

One of the valuable aspects of SOA is the ability to compose and reuse existing services quickly in new ways to support new business processes. Having done this, a question arises regarding how the execution flow of such services can be managed, especially if the business process is long running and one in which the usual guarantees provided by atomic transactions are not appropriate. It is possible to write to the service consumer to handle the orchestration and recovery of all the individual services making up a business process, but this would be a complex undertaking. For example, what happens if the service consumer itself fails?

In response to such needs, some higher-level standards are being developed to facilitate this extended use of SOA services. One example is the Business Process Execution Language (BPEL). BPEL is a high-level language designed to help orchestrate a business process composed of multiple individual services. It provides a means to express such things as when to wait for messages, when to send messages, when to compensate for failed transactions, and so on.

Another area in which a common need arises is for the presentation of existing applications as SOA services. Such applications do not have WSDL interface definitions and are not invoked by SOAP over HTTP messages. A useful function, therefore, is an infrastructure that is interposed between the service consumer and the service provider that provides protocol and data transformation between WSDL, SOAP, HTTP, HTTPS, and the native protocol of the service provider. This transformation function is performed by a component known as a SOA "service adapter."

SOA enterprise service bus

These requirements have spawned the concept of a SOA enterprise service bus (ESB). An ESB can be thought of as middleware infrastructure that acts as a broker between the service consumer and the service provider (see figure 3). In addition to supporting such things as business process orchestration and data transformation, ESBs also provide other higher-level SOA services, such as:

- Service abstraction and content-based routing
- UDDI registry
- Publish or subscribe relationships between service consumers and providers
- Security control and enforcement
- Simplified manageability

Examples of ESB products include Oracle Enterprise Service Bus, IBM WebSphere Enterprise Service Bus, webMethods Ensemble, Sonic ESB from Progress, and the open-source Iona Celtix Enterprise, Apache ServiceMix, OpenESB, Mule ESB products.

Use of ESBs is only really necessary when the SOA model is being applied on a large scale across an enterprise. In this case, the sheer number of services involved makes an ad hoc approach unmanageable, which necessitates a more controlled or "governed" approach. So an ESB may not be necessary when first starting out on the SOA path, when enabling a small set of existing applications. But as the number of services grows and the nature of those services becomes more heterogeneous (that is, the services are implemented in a number of different programming environments and distributed across a number of platforms), then the capabilities provided by an ESB infrastructure become more necessary.

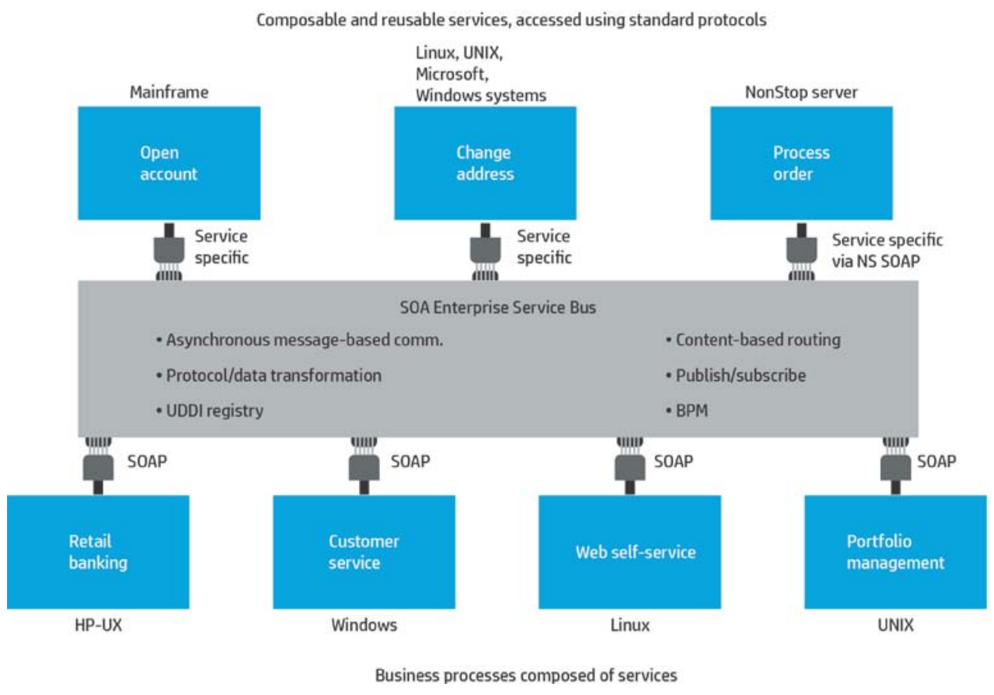
You can integrate services deployed on the Integrity NonStop server with an ESB. ESBs provide the means to "plug in" SOA services, regardless of the implementation. This may be done either by making such SOA services available as Web services directly, or through the use of proxies, which expose the target service as a Web service, and then perform the necessary protocol and message transformation required to invoke the actual service in its native fashion. ESBs provide configuration tools to support this capability. SOA services provided on the Integrity NonStop server may, therefore, be integrated with and accessed using an ESB running on another platform.

SOA and the Integrity NonStop server—an overview

Now that we have described the basic principles and technologies underpinning the SOA model, what does this mean for applications deployed on the Integrity NonStop server?

If it all sounds complicated so far, this section helps show that initial SOA implementations on the NonStop server are actually quite easy to achieve.

Figure 3. SOA ESB


Composable and reusable services, accessed using standard protocols
Mainframe
Linux, UNIX, Microsoft, Windows systems
NonStop server
Open account
Change address
Process order
Service specific
Service specific
Service specific via NS SOAP
SOA Enterprise Service Bus
• Asynchronous message-based comm.
• Protocol/data transformation
• UDDI registry
• Content-based routing
• Publish/subscribe
• BPM
SOAP
SOAP
SOAP
SOAP
Retail banking
Customer service
Web self-service
Portfolio management
HP-UX
Windows
Linux
UNIX
Business processes composed of services

Given that a SOA service may be composed as part of several business processes, it must be able to support a large number of simultaneous service consumers. Likewise, the business processes may implement critical parts of the enterprise; therefore, high levels of availability are required. These SOA service-level requirements of high scalability and availability are tailor made for the Integrity NonStop server, which is built from the ground up to support these attributes. Additional SOA requirements include high performance levels and easy manageability—again, areas where the Integrity NonStop server excels.

Most Integrity NonStop server applications are already aligned with the SOA model because services are the natural way to architect applications on the Integrity NonStop server. For example, “Create Order” or “Fulfill Order” functions can be considered as SOA services, and can be implemented as Pathway server classes, NonStop Processes, NonStop Servlets for JSP, NonStop Tuxedo services, or NonStop CORBA objects. Given this alignment with the SOA model, making existing Integrity NonStop server applications SOA enabled is a relatively straightforward process. As a start, all that is required is to encapsulate the existing application implementation with a standard SOA service interface in WSDL. There is no need to change the application.

SOA and the Integrity NonStop server—product technologies

The Integrity NonStop server supplies a product set that supports the WS-I Basic Profile. Therefore, it is possible to construct and deploy SOA services on the Integrity NonStop server that are fully standards compliant, and which may interoperate with service consumers and service providers running on other platforms that also support the WS-I Basic Profile.

The NonStop server SOA product technologies provide the necessary capabilities for:

- Service access—how remote service consumers access service providers on the Integrity NonStop server
- Service invocation—how service providers are invoked from a service adapter on the Integrity NonStop server
- Service implementation—how the service provider business logic is implemented

HP iTP WebServer software

HP iTP WebServer software provides the HTTP and HTTPS protocol service for all the other SOA components. Built on the HP NonStop Transaction Services/MP (NonStop TS/MP) infrastructure, iTP WebServer software provides a fault-tolerant and scalable container for Web service execution, hosting both NonStop SOAP and NonStop Servlets for JSP components.

HP NonStop SOAP software

HP NonStop SOAP software supports the standard SOAP protocol. The combination of iTP WebServer and NonStop SOAP software provides the standard SOAP over HTTP protocol for invoking SOA services on the Integrity NonStop server. NonStop SOAP software runs as scalable server classes using the iTP WebServer and NonStop TS/MP infrastructure. It provides capabilities that make it very easy to expose existing Pathway servers or NonStop processes as SOA services. This includes a GUI wizard that generates WSDL automatically from Pathway server DDL descriptions, as well as a built-in Pathway service adapter to translate SOAP over HTTP messages to Pathway server invocations. The NonStop SOAP server is fully customizable and can also be used with other types of application services such as NonStop Tuxedo or NonStop CORBA software. An XML parser is included with the NonStop SOAP software product. A SOAP client (NonStop SOAP 4 or gSOAP) for accessing Web services on other platforms through SOAP requests is provided on all site update tapes (SUTs) as part of the NonStop H-series operating system bundle.

HP NonStop Servlets for JavaServer Pages software

HP NonStop Servlets for JSP software is a fortified version of the Apache Tomcat Web container that exhibits NonStop availability and scalability while supporting the standard Java Platform Enterprise Edition (JEE) Servlets and JSP programming models. NonStop Servlets for JSP software runs as scalable server classes using the iTP WebServer and NonStop TS/MP infrastructure. NonStop Servlets for JSP software provides a container for the deployment of Java SOA objects. These can implement the actual business processes themselves, or they can provide a Java service adapter to the actual business processes written to other models. The Java service adapter model is an alternate way of using NonStop SOAP software to SOA, which enables existing applications.

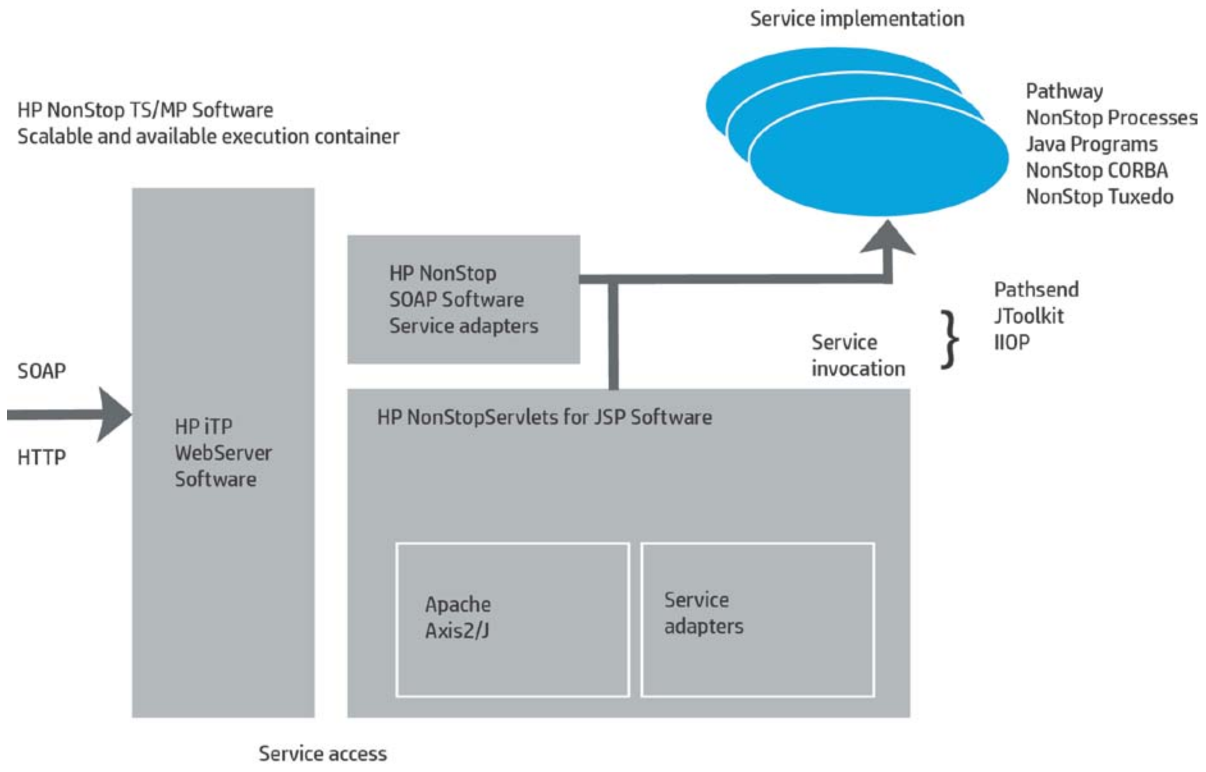
Apache Axis2/Java

Apache Axis2/Java, an open-source product that has been certified for use on the NonStop server, supports a SOAP implementation in Java and a set of tools to simplify the development of SOA services in Java. Axis2 can be used in conjunction with the scalable NonStop Servlets for JSP container to implement either Java service adapters or Java SOA business processes.

JToolkit for NonStop Servers software

JToolkit for NonStop Servers software enables easy access to Pathway servers and Enscribe flat file data from Java-based SOA services.

Figure 4. The Integrity NonStop server SOA product toolset



Summary

From a business perspective, the SOA model is an important tool that can make IT more responsive to changing business needs, thereby improving business agility. The basic SOA principles are designed to preserve and leverage existing IT investment, simplify the development of new application services, and facilitate application interoperability, thereby enabling new business processes to be deployed more quickly and less expensively.

The Integrity NonStop server can play an important role in a SOA architecture as a first-class platform for the provision of SOA services. Using the Integrity NonStop server in this role brings the values of application and data scalability, availability, data integrity, and ease of manageability to SOA services—without requiring special programming.

Increasingly, business applications are deployed across heterogeneous platforms. For this complex IT world, SOA provides a standards-based framework that enables HP Integrity NonStop systems to seamlessly participate in the execution of business processes spanning UNIX, Microsoft, Windows, Linux, and other heterogeneous operating environments.

The Integrity NonStop server platform provides the necessary tools and infrastructure to expose the server applications easily as SOA services in a standard fashion. This both preserves customer investment in existing applications by enabling them to be leveraged and composed as part of new SOA applications, as well as enables new applications to be developed for the Integrity NonStop server that conform to the SOA model.

For more information

The Integrity NonStop server can play an important role in the implementation of SOA business processes.
To know more, visit: hp.com/go/nonstop

Get connected

hp.com/go/getconnected

Current HP driver, support, and security alerts
delivered directly to your desktop

4AA1-0314ENW, Created February 2007; Updated May 2012, Rev. 1