Java Frameworks

- JEE Releases
- JPE (May 1998)
- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)
- J2EE 1.4 (November 11, 2003)
- Java EE 5 (May 11, 2006)
- Java EE 6 (December 10, 2009)
- CDI
- SOA
- ESB and Web Services
- WSDL / SOAP
- SOAP UI
- JAXP (JEE4 2003)
- JAX-RPC (JEE4 2003)
- JAXR (JEE4 2003)
- Axis (2004)
- XFire (2004)
- JAXB (JEE5 2006)
- JAX-WS (JEE5 2006)
- JAX-RS 2006 (JEE5 2006)
- CXF (2008)
- REST (2010)
- WebSocket (2012)
- Spring Services (2002 - present)
- Netty Overview (2011)

JEE Releases

Contents

JPE (May 1998)

Announcement of the JPE (Java Professional Edition) project at Sun.

J2EE 1.2 (December 12, 1999)

List of J2EE 1.2 specifications Developer's Guide.

Technology	Version
JDBC Standard Extension API	2.0
Java Naming and Directory Interface Specification (JNDI)	1.2
RMI-IIOP	1.1
Java Servlet	2.2
JavaServer Pages (JSP)	1.1
Enterprise JavaBeans (EJB)	1.1
Java Message Service API (JMS)	1.0
Java Transaction API (JTA)	1.0
JavaMail API	1.1
JavaBeans Activation Framework (JAF)	1.0

J2EE 1.3 (September 24, 2001)

List of J2EE 1.3 specifications developed under JSR 58 Tutorial

Technology	Version
JDBC Extension	2.0
Java API for XML Processing (JAXP)	1.1
Java Servlet	2.3
JavaServer Pages (JSP)	1.2
JavaServer Pages Standard Tag Library (JSTL)	1.0
Enterprise JavaBeans (EJB)	2.0
J2EE Connector Architecture	1.0
Java Message Service API (JMS)	1.0
Java Transaction API (JTA)	1.0
JavaMail API	1.2
JavaBeans Activation Framework (JAF)	1.0
Java Authentication and Authorization Service (JAAS)	1.0

J2EE 1.4 (November 11, 2003)

List of J2EE 1.4 specifications developed under JSR 151 Tutorial

Technology	Version	JSR

Web Services Technologies:		
Web Services for J2EE 1.1	1.0	
Java API for XML Processing (JAXP)	1.2	
Java API for XML-based RPC (JAX-RPC)	1.1	
Java API for XML Registries (JAXR)	1.0	
Web Application Technologies:		
Java Servlet	2.4	JSR154
JavaServer Pages (JSP)	2.0	JSR152
JavaServer Pages Standard Tag Library (JSTL)	1.1	JSR52
JavaServer Faces (JSF)	1.1	JSR127
Enterprise Application Technologies:		
Enterprise JavaBeans (EJB)	2.1	JSR153
J2EE Connector Architecture	1.5	JSR112
Java Message Service API (JMS)	1.1	
Java Transaction API (JTA)	1.0	
JavaMail API	1.3	
JavaBeans Activation Framework (JAF)	1.0	
Management and Security Technologies:		
Java Authorization Service Provider Contract for Containers (JACC)	1.0	
Java Management Extensions (JMX)	1.2	
Enterprise Edition Management API	1.0	
Enterprise Edition Deployment API	1.1	

Java EE 5 (May 11, 2006)

List of Java EE 5 specifications developed under JSR244 Tutorial

Technology	Version	JSR
Web Services Technologies:		
Web Services	1.2	JSR109
Java API for XML-Based Web Services (JAX-WS)	2.0	JSR224
Java Architecture for XML Binding (JAXB)	2.0	JSR222
Web Service Metadata for the Java Platform	2.0	JSR181
Java API for XML-Based RPC (JAX-RPC)	1.1	JSR101
Java API for XML Registries (JAXR)	1.0	JSR93
SOAP with Attachments API for Java (SAAJ)	1.3	JSR67
Streaming API for XML (StAX)	1.0	JSR173
Web Application Technologies:		
Java Servlet	2.5	JSR154

JavaServer Faces (JSF)	1.2	JSR252
JavaServer Pages (JSP)	2.1	JSR245
JavaServer Pages Standard Tag Library (JSTL)	1.2	JSR52
Debugging Support for Other Languages	1.0	JSR45
Enterprise Application Technologies:		
Enterprise JavaBeans (EJB)	3.0	JSR220
Java Persistence API (JPA)	1.0	JSR220
Java EE Connector Architecture	1.5	JSR112
Common Annotations for the Java Platform	1.0	JSR250
Java Message Service API (JMS)	1.1	JSR914
Java Transaction API (JTA)	1.1	JSR907
JavaMail API	1.4	JSR919
JavaBeans Activation Framework (JAF)	1.1	JSR925
Management and Security Technologies:		
Java Authorization Service Provider Contract for Containers (JACC)	1.1	JSR115
J2EE Application Deployment	1.2	JSR88
J2EE Management	1.1	JSR77

Java EE 6 (December 10, 2009)

Java EE 6 introduced the concept of profile, which represents a configuration of the platform suited to a particular class of applications. The Web Profile offers a complete stack, with technologies addressing presentation and state management (JavaServer Faces, JavaServer Pages), core web container functionality (Servlet), business logic (Enterprise JavaBeans Lite), transactions (Java Transaction API), persistence (Java Persistence API) and more.

List of Java EE 6 specifications developed under JSR 316 Tutorial

Technology	Version	JSR	Included in Web Profile
Web Services Technologies:			
Java API for RESTful Web Services (JAX-RS)	1.1	JSR311	
Web Services	1.3	JSR109	
Java API for XML-Based Web Services (JAX-WS)	2.2	JSR224	
Java Architecture for XML Binding (JAXB)	2.2	JSR222	
Web Services Metadata for the Java Platform	2.1	JSR181	
Java API for XML-based RPC (JAX-RPC)	1.1	JSR101	
Java APIs for XML Messaging (JAXM)	1.3	JSR67	
Java API for XML Registries (JAXR)	1.0	JSR93	
Web Application Technologies:			
Java Servlet	3.0	JSR315	
JavaServer Faces (JSF)	2.0	JSR314	
JavaServer Pages (JSP)	2.2	JSR245	
Expression Language (EL)	2.2	JSR245	
JavaServer Pages Standard Tag Library (JSTL)	1.2	JSR52	

Debugging Support for Other Languages	1.0	JSR45	
Enterprise Application Technologies:			
Enterprise JavaBeans (EJB)	3.1	JSR318	Lite
Java Persistence API (JPA)	2.0	JSR317	
Contexts and Dependency Injection for Java	1.0	JSR299	
Dependency Injection for Java	1.0	JSR330	
Bean Validation	1.0	JSR303	
Managed Beans	1.0	JSR316	
Interceptors	1.1	JSR318	
Java EE Connector Architecture	1.6	JSR322	
Common Annotations for the Java Platform	1.1	JSR250	
Java Message Service API (JMS)	1.1	JSR914	
Java Transaction API (JTA)	1.1	JSR907	
JavaMail API	1.4	JSR919	
Management and Security Technologies:			
Java Authentication Service Provider Interface for Containers (JASPIC)	1.0	JSR196	
Java Authorization Service Provider Contract for Containers (JACC)	1.4	JSR115	
Java EE Application Deployment	1.2	JSR88	
J2EE Management	1.1	JSR77	

CDI

Overview of CDI

The most fundamental services provided by CDI are as follows:

- **Contexts**: The ability to bind the lifecycle and interactions of stateful components to well-defined but extensible lifecycle contexts
- **Dependency injection**: The ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject

In addition, CDI provides the following services:

- Integration with the Expression Language (EL), which allows any component to be used directly within a JavaServer Faces page or a JavaServer Pages page
- The ability to decorate injected components
- The ability to associate interceptors with components using typesafe interceptor bindings
- An event-notification model
- A web conversation scope in addition to the three standard scopes (request, session, and application) defined by the Java Servlet specification
- A complete Service Provider Interface (SPI) that allows third-party frameworks to integrate cleanly in the Java EE 6 environment

SOA

Definitions

The OASIS group[4] and the Open Group[5] have both created formal definitions. OASIS defines SOA as:

> *A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*

The Open Group's definition is:

> *Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.*
>
> *A service:*
>
> *Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)Is self-containedMay be composed of other servicesIs a "black box" to consumers of the service*

Overview

Services are **unassociated**, loosely coupled units of functionality that are self-contained. Each service implements at least one action, such as submitting an online application for an account, retrieving an online bank statement or modifying an online booking or airline ticket order. Within an SOA, services use defined protocols that describe how services pass and parse messages using description metadata, which in sufficient details describes not only the characteristics of these services, but also the data that drives them. Programmers have made extensive use of XML in SOA to structure data that they wrap in a nearly exhaustive description-container. Analogously, the Web Services Description Language (WSDL) typically describes the services themselves, while SOAP (originally Simple Object Access Protocol) describes the communications protocols. SOA depends on data and services that are described by metadata that should meet the following two criteria:

1. The metadata should be provided in a form that software systems can use to configure dynamically by discovery and incorporation of defined services, and also to maintain coherence and integrity. For example, metadata could be used by other applications, like a catalogue, to perform auto discovery of services without modifying the functional contract of a service.
2. The metadata should be provided in a form that system designers can understand and manage with a reasonable expenditure of cost and effort.

The purpose of SOA is to allow users to combine together fairly large chunks of functionality to form *ad hoc* applications built almost entirely from existing software services. The larger the chunks, the fewer the interfaces required to implement any given set of functionality; however, very large chunks of functionality may not prove sufficiently granular for easy reuse. Each interface brings with it some amount of processing overhead, so there is a performance consideration in choosing the granularity of services.

SOA as an architecture relies on service-orientation as its fundamental design principle. If a service presents a simple interface that abstracts away its underlying complexity, then users can access independent services without knowledge of the service's platform implementation.[6]

SOA framework

SOA-based solutions endeavour to enable business objectives while building an enterprise-quality system. SOA architecture is viewed as five horizontal layers:[7]

1. Consumer Interface Layer – These are GUI for end users or apps accessing apps/service interfaces.
2. Business Process Layer – These are choreographed services representing business use-cases in terms of applications.
3. Services – Services are consolidated together for whole-enterprise in-service inventory.
4. Service Components – The components used to build the services, such as functional and technical libraries, technological interfaces etc.
5. Operational Systems – This layer contains the data models, enterprise data repository, technological platforms etc.

There are four cross-cutting vertical layers, each of which are applied to and supported by each of the following horizontal layers:

1. Integration Layer – starts with platform integration (protocols support), data integration, service integration, application integration, leading to enterprise application integration supporting B2B and B2C.
2. Quality of Service – Security, availability, performance etc. constitute the quality of service parameters which are configured based on required SLAs, OLAs.
3. Informational – provide business information.
4. Governance – IT strategy is governed to each horizontal layer to achieve required operating and capability model.

ESB and Web Services

An ESB, as we know, is an architectural pattern that enables you to optimize the distribution of information between different types of applications across multiple locations. The ESB pattern is founded on and unifies message-oriented, event-driven and service-oriented approaches to integration.

Providing connectivity and integration for Web services-focused applications and services, the core characteristics of an ESB provide:

- Standards-based application integration
- Support for Web services, message-based transport and publish-and-subscribe (event-based) integration
- Transformation
- Intelligent routing

Still, it is important to understand the following points about ESBs:

- The term enterprise does not necessarily have to encompass the whole of an organization. One of the attractions of an ESB is that you can start small, perhaps with only two to three physical instances and expand to fit evolving business circumstances.

- The term bus is used to convey the notion of information being carried between originators and receivers, using different communications models and data formats, to many different destinations.
- The bus provides a common backbone through which applications can interoperate.
- An ESB should possess some degree of programming intelligence to determine routing or persistence, or to implement rules or content processing.

An ESB introduces new options for interoperation and helps enable information to flow to the people who need it, when they need it. In this way, an ESB can improve the responsiveness and accuracy of decision making.

Ideally, an ESB should describe (at a high, logical level) what is going on within an enterprise, or subset of an enterprise. This description should be visual or graphical so that both the available resources and the flows (from sources to destinations) can be represented and depicted. Besides describing the flows between the various nodes (sources, destinations, databases, local ESB processing, etc.), an ESB should enable you to lay on top of these resources the particular constraints associated with how flows are actually deployed and used.

The more an ESB can describe, the more capabilities can be automated. If applicable resources are described, network bandwidths set, systems located and transaction boundaries defined, you can then determine where potential problems might occur – and how these problem areas might be avoided or resolved. There should be a clear distinction between what is to be achieved, and how it is to be achieved.

In one sense, this approach envisions applications as existing on the edge of an ESB. The first step you should take is to integrate each application with an ESB. Having done this for several applications, you can then create new, logical combinations of applications out of the definitions of existing ones. An ESB's flow capability effectively enables new combinations to be derived and implemented without opening up or rewriting the source or destination applications themselves.

In an ideal ESB environment, you should be able to define the applications and flows that shape your organization. You should also be able to add new combinations of applications and flows – or modify existing ones – easily.

Web services and ESBs

Combining Web services standards with an ESB can potentially deliver the broadest connectivity between systems. An ESB supports Web services with more established application-integration techniques, enabling the power of the new to be combined with the reach of the old.

Fundamentally, early Web-services standards were principally concerned with the format of what was being carried and less about the mechanism of the carriage. Web services were essentially remote procedure calls or one-way call formats. These services did not indicate how the data would be carried to the other end. You could assume that it would be by HTTP, but there was no discussion about different kinds of quality of service or other attributes that might be applicable.

The presumption was that any connection would be synchronous and that any asynchronous properties would have to be built, as required, on top of the synchronous connection by using a work-around. Web services become interesting at several levels – particularly when Web services start to interact with each other. However, you must remember that not every application is written as a Web service. Valuable legacy applications will almost certainly be in use. Product extensions like IBM CICS Web Services Bridge can help in these situations by enabling Web services and legacy applications to connect. These situations are where the ESB concept can come into focus.

An ESB offers a means by which modern Web-services applications and valuable legacy applications can make the most of each other and developers don't have to create new applications to carry out the functions of the familiar ones that couldn't be connected.

What an ESB adds is a common intermediary point or points where the rigidities of individual point-to-point connections can be avoided. In this instance, an ESB is effectively a switch for connecting multiple Web-services applications and legacy applications – or even other resources, like databases or application servers.

ESB: Is it an architecture or a product?

There has been much discussion and debate in the industry as to whether an ESB is a product or an architecture. However, there is little argument when it comes to providing connectivity and integration for Web services-focused applications and services that an ESB is the recommended path.

The answer to the question on architecture versus product really lies within the organization itself. Inherent in the term SOA is the word architecture and a critical component of a successful deployment is an ESB. For organizations that may not yet need an extensive integration of services yet want to provide universal connectivity and data transformation for applications whether or not they comply with standards, they may opt for a plug-in ESB product.

You may ask the architecture versus product question another way: are you looking to build an enterprise-wide strategy or build a bridge for connecting disparate applications?

From a Web services point of view, what advantage does an ESB offer?

Web services-based applications can be deployed to exploit an ESB with little or no modification to the application. A benefit of Web services in an ESB context is that they enable you to take an architectural approach to decomposing IT solutions into pieces that can be reused. This helps increase efficiency because it makes the delivery of agile information systems that are responsive to changing business demands easier.

Similarly, loosely coupled, federated reuse of existing assets is made possible by combining the standardization inherent in Web services with the flexibility inherent in an ESB. The ESB offers the capability to support a wide range of devices, from PDAs to cell phones to telemetry devices to mainframes. The ESB can also support applications, from new Web services-based applications to legacy applications and enables them to work together as peers in a distributed environment.

A developer can write new applications for an ESB using the Web-services development infrastructure products that deliver the following benefits:

- Provide the means to define the SOAP message formats
- Help you easily enable disparate consumers and providers of services to interact properly
- Enable the application programmer to see the service as a standard method definition and then call that service using a standard remote method call. The benefit of this is that the programmer does not need to be involved in details of format or transport.

Conclusion

The adoption and proliferation of Web services standards will continue to grow and be in high demand. This will lead to a greater need for the conversion and wrapping capabilities of an ESB and underscores the need for the flexibility of an ESB in performing the transformations, routing and interconnection between legacy and new, Web-services based applications is unlikely to disappear.

We can expect even closer integration of Web services and other service-oriented architectures with ESBs, as well as closer integration with application servers. This tight integration has positive benefits for application server-based applications. You can use an ESB for Web-services delivery, whether as service consumer, provider or intermediate, while also improving implementation efficiency.

For these reasons, adopting Web services in conjunction with an ESB can help preserve your software and hardware investments. New applications can be written more easily using Web services because of modern development tooling and its improved usability. At the same time, these new capabilities can be combined, or integrated, with existing IT infrastructure and applications. As a result, an ESB can enable you to consider exploiting past investments in infrastructure and applications. Similarly, new investments in Web services-based applications are preserved as your organization's infrastructure evolves.

WSDL / SOAP

WDSL

The Web Services Description Language (WSDL /wz dl/) is an XML-based interface definition language that is used for describing the functionality offered by a web service. The acronym is also used for any specific WSDL description of a web service (also referred to as a WSDL file), which provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a purpose that corresponds roughly to that of a method signature in a programming language.

SOAP

SOAP, originally an acronym for Simple Object Access protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML Information Set for its message format, and relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP UI

SoapUI is a free and open source cross-platform Functional Testing solution. With an easy-to-use graphical interface, and enterprise-class features, SoapUI allows you to easily and rapidly create and execute automated functional, regression, compliance, and load tests. In a single test environment, SoapUI provides complete test coverage and supports all the standard protocols and technologies. There are simply no limits to what you can do with your tests. Meet SoapUI, the world's most complete testing tool!

JAXP (JEE4 2003)

The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language. JAXP leverages the parser standards Simple API for XML Parsing (SAX) and Document Object Model (DOM) so that you can choose to parse your data as a stream of events or to build an object representation of it. JAXP also supports the Extensible Stylesheet Language Transformations (XSLT) standard, giving you control over the presentation of the data and enabling you to convert the data to other XML documents or to other formats, such as HTML. JAXP also provides namespace support, allowing you to work with DTDs that might otherwise have naming conflicts. Finally, as of version 1.4, JAXP implements the Streaming API for XML (StAX) standard.

Designed to be flexible, JAXP allows you to use any XML-compliant parser from within your application. It does this with what is called a pluggability layer, which lets you plug in an implementation of the SAX or DOM API. The pluggability layer also allows you to plug in an XSL processor, letting you control how your XML data is displayed.

Released JEE4 2003

Deprecated in JEE6 2009

JAX-RPC (JEE4 2003)

Java API for XML-based RPC (**JAX-RPC**) allows a Java application to invoke a Java-based Web service with a known description while still being consistent with its WSDL description. JAX-RPC is one of the Java XML programming APIs. It can be seen as Java RMIs over Web

services. JAX-RPC 2.0 was renamed JAX-WS 2.0 (Java API for XML Web Services). JAX-RPC 1 is deprecated with Java EE 6.[1] The JAX-RPC service utilizes W3C (World Wide Web Consortium) standards like WSDL or Web Service Description Language.[2]

It works as follows:

1. A Java program executes a method on a stub (local object representing the remote service)
2. The stub executes routines in the JAX-RPC Runtime System (RS)
3. The RS converts the remote method invocation into a SOAP message
4. The RS transmits the message as an HTTP request

The advantage of such a method is that it allows the Web service to be implemented at server-side as a Servlet or EJB container. Thus, Servlet or EJB applications are made available through Web services.

Example tool wsdl2java java2wsdl

Released JEE4 2003

Deprecated in JEE6 2-009

JAXR (JEE4 2003)

Java API for XML Registries (JAXR) defines a standard API for Java platform applications to access and programmatically interact with various kinds of metadata registries. JAXR is one of the Java XML programming APIs. The JAXR API was developed under the Java Community Process as JSR 93.

JAXR provides a uniform and standard Java API for accessing different kinds of XML-based metadata registry. Current implementations of JAXR support ebXML Registry version 2.0, and UDDI version 2.0. More such registries could be defined in the future. JAXR provides an API for the clients to interact with XML registries and a service provider interface (SPI) for the registry providers so they can plug in their registry implementations. The JAXR API insulates application code from the underlying registry mechanism. When writing a JAXR based client to browse or populate a registry, the code does not have to change if the registry changes, for instance from UDDI to ebXML.

Released JEE 4 2003

Deprecated in JEE6 2009

Axis (2004)

Apache Axis2™ is a Web Services / SOAP / WSDL engine, the successor to the widely used Apache Axis SOAP stack. There are two implementations of the Apache Axis2 Web services engine - Apache Axis2/Java and Apache Axis2/C. While you will find all the information on Apache Axis2/Java here, you can visit the **Apache**

Axis2/C Web site for Axis2/C implementation information.

XFire (2004)

XFire is a leading open source web service creation framework that greatly simplifies the creation of non-trivial Java web services. You can find the latest downloads, and a lot of information on XFire, at http://xfire.codehaus.org. Technically, XFire can create ready-to-deploy WAR files of web services, directly from your POJOs, that can be deployed on any Servlet 2.4 compliant servlet container such as Tomcat 5 or Jetty. As well as servlets based on Tomcat, XFire supports other lightweight application container technologies, including Spring, PicoContainer, and Plexus. XFire can be easily embedded inside a Java application providing web service clientor server functionality

JAXB (JEE5 2006)

The goal of the JAXB project is to develop and evolve the code base for the Reference Implementation (RI) of JAXB, the Java Architecture for XML Binding. The JAXB specification is developed through the Java Community Process following the process described at jcp.org. This process involves an Expert Group with a lead that is responsible for delivering the specification, a reference implementation (RI) and a Technology Compatibility Kit (TCK). The primary goal of an RI is to support the development of the specification and to validate it. Specific RIs can have additional goals; the JAXB RI is a *production-quality implementation* that is used directly in a number of products by Oracle and other vendors.

- jaxb 1.0 March 2003

Released JEE5 2006

JAX-WS (JEE5 2006)

The JAX-WS 2.2 specification JSR 224 defines a standard Java- to-WSDL mapping which determines how WSDL operations are bound to Java methods when a SOAP message invokes a WSDL operation. This Java-to-WSDL mapping determines which Java method gets invoked and how that SOAP message is mapped to the method's parameters.

This mapping also determines how the method's return value gets mapped to the SOAP response.

JAX-WS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints. It is part of the Java Web Services Development Pack. JAX-WS can be used in Java SE starting with version 6.[1] JAX-WS 2.0 replaced the JAX-RPC API in Java Platform, Enterprise Edition 5 which leans more towards document style Web Services.

This API provides the core of Project Metro, inside the GlassFish open-source Application Server community of Oracle Corporation.

JAX-WS also is one of the foundations of WSIT.

Released JEE5 2006

JAX-RS 2006 (JEE5 2006)

JAX-RS: Java API for RESTful Web Services (**JAX-RS**) is a Java programming language API that provides support in creating web services ac cording to the Representational State Transfer (REST) architectural pattern.[1] JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

From version 1.1 on, JAX-RS is an official part of Java EE 6. A notable feature of being an official part of Java EE is that no configuration is necessary to start using JAX-RS. For non-Java EE 6 environments a (small) entry in the web.xml deployment descriptor is required.

Released JEE5 2006

CXF (2008)

Apache CXF is an open-source, fully featured Web services framework. It originated as the combination of two open-source projects: Celtix developed by IONA Technologies (acquired by Progress Software in 2008) and XFire developed by a team hosted at Codehaus. These two projects were combined by people working together at the Apache Software Foundation and the new name **CXF** was derived by combining "*Celtix* " and "*XFire*".

The CXF key design considerations include:

- Clean separation of front-ends, like JAX-WS, from the core code.
- Simplicity with, for instance, the creation of clients and endpoints without annotations.
- High performance with minimum computational overhead.
- Embeddable Web service component: example embeddings include Spring Framework and Geronimo.

Most widely used Web Services Standard Now; Improvement over AXIS2, which is now gradually being replaced by Apache CXF
Intuitive & Easy to Use (less coding required as compared to AXIS2)
Clean separation of front-ends, like JAX-WS, from the core code
Fully compliant with JAX-WS, JAX-RS & others
Best Performance across all available framework with minimum computation overhead
Supports wide variety of front-end models
Supports both JAX-WS & JAX-RS (for Restful Services)
Supports JBI & SDO (not supported in AXIS2)
Compatible with Spring Framework
Released around 2009

REST (2010)

REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. 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. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.[3]

Client and Server Caching

WebSocket (2012)

WebSocket is a protocol providing full-duplex communication channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.

WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.[1] The WebSocket protocol makes more interaction between a browser and a website possible, facilitating live content and the creation of real-time games. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server. The communications are done over TCP port number 80, which is of benefit for those environments which block non-web Internet connections using a firewall. Similar two-way browser-server communications have been achieved in non-standardized ways using stop-gap technologies such as Comet.

The WebSocket protocol is currently supported in most major browsers including Google Chrome, Internet Explorer, Firefox, Safari and Opera. WebSocket also requires web applications on the server to support it.

Spring Services (2002 - present)

The **Spring Framework** is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, replacement for, or even addition to the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

Version history

The first version was written by Rod Johnson, who released the framework with the publication of his book *Expert One-on-One J2EE Design and Development* in October 2002. The framework was first released under the Apache 2.0 license in June 2003. The first milestone release, 1.0, was released in March 2004, with further milestone releases in September 2004 and March 2005. The Spring 1.2.6 framework won a Jolt productivity award and a JAX Innovation Award in 2006.[2][3] Spring 2.0 was released in October 2006, Spring 2.5 in November 2007, Spring 3.0 in December 2009, Spring 3.1 in December 2011, and Spring 3.2.5 in November 2013.[4] The current version is Spring Framework 4.0, which was released in December 2013.[5] Notable improvements in Spring 4.0 include support for Java SE 8, Groovy 2, some aspects of Java EE7, and WebSocket.

Modules

The Spring Framework includes several modules that provide a range of services:

- **Spring Core Container**: This is the base module of Spring and provides spring containers (BeanFactory and ApplicationContext).[6]
- **Aspect-oriented programming**: enables implementing cross-cutting concerns.
- **Authentication and authorization**: configurable security processes that support a range of standards, protocols, tools and practices via the Spring Security sub-project (formerly *Acegi Security System for Spring*).
- **Convention over configuration**: a rapid application development solution for Spring-based enterprise applications is offered in the Spring Roo module
- **Data access**: working with relational database management systems on the Java platform using JDBC and object-relational mapping tools and with NoSQL databases
- **Inversion of control container**: configuration of application components and lifecycle management of Java objects, done mainly via dependency injection
- **Messaging**: configurative registration of message listener objects for transparent message-consumption from message queues via JMS, improvement of message sending over standard JMS APIs
- **Model–view–controller**: an HTTP- and servlet-based framework providing hooks for extension and customization for web applications and RESTful Web services.
- **Remote access framework**: configurative RPC-style marshalling of Java objects over networks supporting RMI, CORBA and HTTP-based protocols including Web services (SOAP)
- **Transaction management**: unifies several transaction management APIs and coordinates transactions for Java objects
- **Remote management**: configurative exposure and management of Java objects for local or remote configuration via JMX
- **Testing**: support classes for writing unit tests and integration tests

Netty Overview (2011)

Writing network based Applications can be challenging without the right abstraction. Netty provides such an abstraction. Netty simplifies network programming while still providing you with all the power and flexibility needed.

Where you would have implemented everything by yourself before, Netty provides you with a rich toolkit and so will boost your productivity. You not need to be an network programming expert to use Netty the same way as you do not need to be a mechanic if you drive a car. Think about how painful it would be if you would need to assemble the whole car by yourself before you can drive it. Well some people would call it fun but for most it wouldn't be. The same kind of analogy is true when using Netty. You also not need to assemble your own framework out of the provided „low-level-API“, which is bundled with Java itself to write your network based application. You can just grab Netty and get your hands dirty right away, without the need to handle the complexity.

When in 2011, Netty founder Trustin Lee left Red Hat to join Twitter the project became independent of any company in an effort to simplify contributions to it. Both Red Hat and Twitter use Netty, so it should come as no surprise that these two companies are the biggest contributors to

Netty at the time of writing. Adoption of Netty continues to increase, as does the number of individual contributors. The community of Netty users is active and the project remains vibrant.