

J2EE Connector Architecture White Paper

Integrating Java applications with existing Enterprise Applications

Executive Summary

The [J2EE Connector Architecture](#), part of [Java 2 Platform, Enterprise Edition \(J2EE\) 1.3](#), specifies a standard architecture for accessing resources in diverse Enterprise Information Systems (EIS). These may include ERP systems such as SAP R/3, mainframe transaction processing systems such as IBM CICS, legacy applications and non-relational database systems.

Today, the [JDBC Data Access API](#) provides easy integration with relational database systems for Java applications. In a similar manner, the Connector Architecture simplifies integration of Java applications with heterogeneous EIS systems.

This paper describes the 1.0 version of the Connector Architecture specification at a high level, including:

- System contracts defined between the J2EE platform-based application server and the EIS resource adapter, providing security, connection pooling, and transaction management facilities.
- Common Client Interface (CCI) between the EIS resource adapter and application components or tools.
- Packaging and deployment of resource adapters in an application server environment.

The Connector Architecture Specification document contains detailed information on the architecture. The specification and the latest information on the Connector Architecture can be found at <http://java.sun.com/j2ee/connector>.

Introduction

Most companies have enormous investments in Enterprise Information Systems (EISs) such as ERP systems, legacy systems, mainframe database and transaction processing systems. Today, leveraging these systems as part of a web-based, multi-tiered application is challenging. EIS vendors provide proprietary interfaces, with varying levels of support for enterprise application integration. Application server vendors have to build and maintain separate interfaces for different supported EISs, and application developers need to manage the system-level issues of security, transactions and connection pooling within the applications themselves.

Challenges in EIS integration

Integration with EISs presents many challenges.

- The back end EISs are complex and heterogeneous. The application programming models vary widely between these systems, increasing the complexity and effort of application integration.
- Application development tools that can simplify these integration efforts are critical.
- Transaction and security management add complexity to integration with back-end EIS systems.
- The web-based architecture requires significant scalability in terms of the potential number of clients that can access enterprise applications.

J2EE Platform and Connector Architecture

The Connector Architecture addresses these challenges directly. The J2EE platform provides a reusable component model, using [Enterprise JavaBeans](#) and [JavaServer Pages](#) technologies to build and deploy multi-tier applications that are platform and vendor-independent. The J2EE platform shares the "Write Once, Run Anywhere" approach of the Java platform, and has significant industry support.

The Connector Architecture adds simplified EIS integration to this platform. The goal is to leverage the strengths of the J2EE platform -- including component models, transaction and security infrastructures -- to address the challenges of EIS integration.

The Connector Architecture defines a common interface between application servers and EIS systems, implemented in EIS-specific resource adapters that plug into application servers. The result is simplified enterprise application integration, using a scalable, standard architecture that leverages the benefits of the J2EE platform.

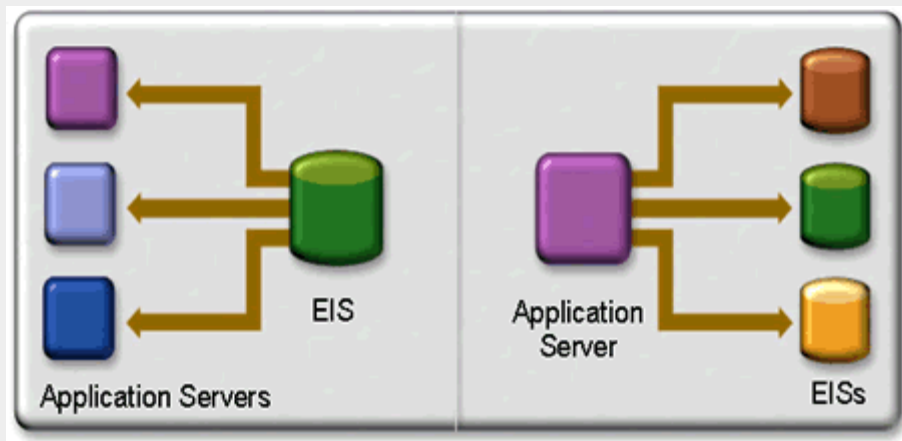


Figure: Using the Connector Architecture, each EIS only writes one Connector Architecture-compliant resource adapter, and each application server extends its system once to support integration with any number of EIS resource adapters and underlying EISs.

Developing a standard contract between application components, application servers and EISs reduces the overall scope of the integration effort. This delivers benefits for the entire Java development community:

EIS vendors only need to create one, open interface (implemented in the resource adapter) to the EIS. This resource adapter can be used with any compliant J2EE application server, and provides a standard interface for tool and Enterprise Application Integration (EAI) vendors. Maintaining a single interface reduces the development effort for the EIS vendor, who today must build point solutions targeted at individual vendors and certify individual systems for compliance.

Application Servers vendors (vendors of any compliant J2EE servers) only need to extend their systems once to support the system contracts defined by the Connector Architecture. Then they can simply plug in multiple resource adapters to extend the server, supporting integration with multiple EISs without any EIS-specific system-level programming.

Enterprise Application Integration (EAI) and development tool vendors use the Common Client Interface (CCI) to simplify access to EIS resources with a standard application-level interface.

Application component developers are shielded from the complexity of transaction, connection and security concerns when accessing data or functions in EISs and can concentrate instead on developing business and application logic.

The Connector Architecture is the product of the Java Community Process program, with the contributions of a wide range of tool, server, and EIS vendors.

Connector Architecture Overview

The Connector Architecture is implemented in an application server and an EIS-specific resource adapter. A resource adapter is a system library specific to an EIS and provides connectivity to the EIS. A resource adapter is analogous to a JDBC driver. The interface between a resource adapter and the EIS is specific to the underlying EIS; it can be a native interface.

The Connector Architecture has three main components:

- System level contracts between the resource adapter and the application server.
- Common Client Interface that provides a client API for Java applications and development tools to access the resource adapter.

Standard packaging and deployment facility for resource adapters.

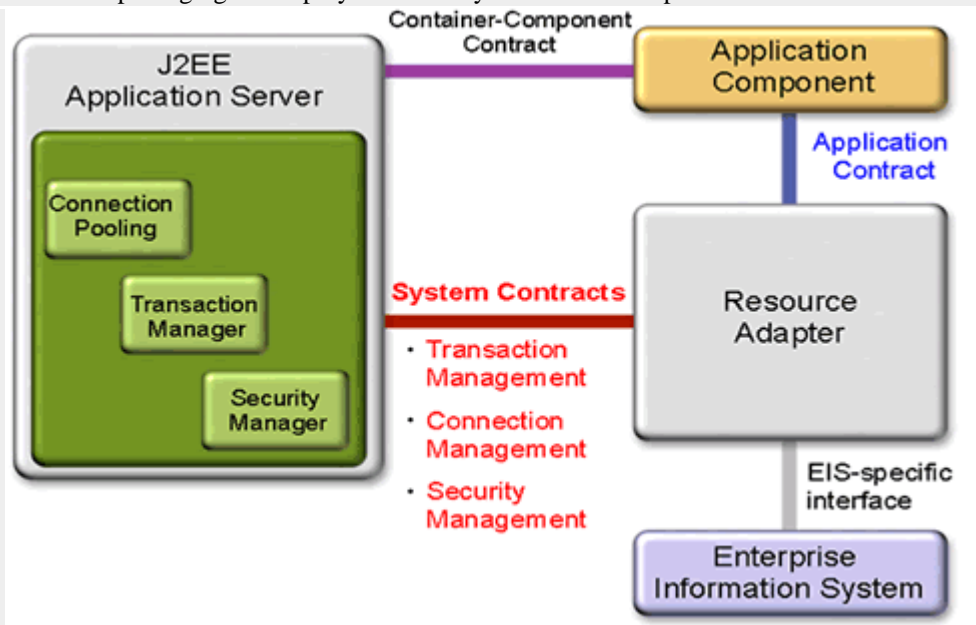


Figure: The Connector Architecture defines system contracts between the application server and the resource adapter. It also defines a client API between the resource adapter and application components. These contracts are described further in this paper.

The containers in an application server can be web containers (hosting JSP pages and Servlets) and EJB containers. The application server provides a set of services in an implementation-specific way. These services include transaction manager, security services manager and connection pooling mechanism. The Connector Architecture does not define how an application server implements these different services.

The application server vendor extends the server to support the system contracts defined by the Connector Architecture. The system contracts can be considered as a Service Provider Interface (SPI). The SPI provides a standard way for extending a container to support connectivity to multiple EISs.

The EIS vendor (or a third party ISV) creates a resource adapter for the EIS, using an EIS-specific interface to interact with the EIS itself and supporting the system contracts for the application server. In addition, the resource adapter provides a client API called the Common Client Interface, or CCI, defined by the Connector Architecture. Application development tools or application components use the Common Client Interface to interact with the resource adapter directly.

The resource adapter runs in the application server's address space and manages access to the EIS resources. A Connector Architecture-compliant resource adapter will work with any compliant J2EE server.

System-Level Contracts

The 1.0 version of the Connector Architecture provides the following system contracts, to be implemented by the resource adapter and the application server:

- Connection management
- Transaction management
- Security management

For the 1.0 release these contracts cover the most pressing concerns for enterprise application integration: transactions, security, and scalability. In future versions of the specification, the system contracts will be extended to include support for [Java Message Service \(JMS\)](#) pluggability and thread management. The JMS pluggability will add support for asynchronous message-based communication.

The following sections offer brief overviews of these contracts.

Connection Management Contract

A connection to an EIS is an expensive system resource. To support scalable applications, an application server needs to pool connections to the underlying EISs. This connection pooling mechanism should be transparent to applications accessing the underlying EIS, simplifying application development.

The Connection Management contract supports connection pooling and management, optimizing application performance and increasing scalability. The Connection Management contract is defined between an application server and a resource adapter. It provides support for an application server to implement its connection pooling facility. The application server structures and implements its connection pool in an implementation specific way - the pool can be very primitive or advanced depending on the quality of services offered by the application server.

The application server uses the connection management contract to:

- create new connections to an EIS
- configure connection factories in the JNDI namespace
- find the right connection from an existing set of pooled connections

The connection management contract enables an application server to hook in its services, such as transaction management and security management.

Transaction Management Contract

Transactional access to EISs is an important requirement for business applications. The Connector Architecture supports the concept of transactions - a number of operations that must be committed together or not at all for the data to remain consistent and to maintain data integrity.

In many cases, a transaction (termed local transaction) is limited in scope to a single EIS system, and the EIS resource manager itself manages such transaction. While an XA transaction (or global transaction) can span multiple resource managers. This form of transaction requires transaction coordination by an external transaction manager, typically bundled with an application server. A transaction manager uses a two-phase commit protocol to manage a transaction that spans multiple resource managers (EISs). It uses one-phase commit optimization if only one resource manager is participating in an XA transaction.

The connector architecture defines a transaction management contract between an application server and a resource adapter (and its underlying resource manager). The transaction management contract extends the connection management contract and provides support for management of both local and XA transactions. The transaction management contract has two parts, depending on the type of transaction.

- JTA XAResource based contract between a transaction manager and an EIS resource manager
- Local transaction management contract

These contracts enable an application server to provide the infrastructure and runtime environment for transaction management. Application components rely on this transaction infrastructure to support the component-level transaction model.

Because EIS implementations are so varied, the transactional support must be very flexible. The Connector Architecture imposes no requirements on the EIS for transaction management. Depending on the implementation of transactions within the EIS, a resource adapter may provide:

- No transaction support at all - this is typical of legacy applications and many back-end systems.
- Support for only local transactions
- Support for both local and XA transactions

An application server is required to support all three levels of transactions. This ensures that application servers can support EISs at different transaction levels.

Security Contract

It is critical that an enterprise be able to depend on the information in its EIS for its business activities. Any loss or inaccuracy of information or any unauthorized access to the EIS can be extremely costly to an enterprise. There are mechanisms that can be used to protect an EIS against such security threats, including:

Identification and authentication of principals (human users) to verify they are who they claim to be.

Authorization and access control to determine whether a principal is allowed to access an application server and/or an EIS.

Security of communication between an application server and an EIS. Communication over insecure links can be protected using a protocol (for example, Kerberos) that provides authentication, integrity, and confidentiality services. Communication can also be protected by using a secure links protocol (for example, SSL).

The Connector Architecture extends the J2EE security model to include support for secure connectivity to EISs.

The security management contract is defined to be independent of security mechanisms and technologies. This enables application servers and EISs with different levels of support for security technology to support the security contract. For example, the security management contract can support basic user-password based authentication or a Kerberos-based end-to-end security environment. It can also support EIS-specific security mechanisms.

EIS Sign-on

Creating a new physical connection requires a sign-on to an EIS. Changing the security context on an existing physical connection can also require EIS sign-on; the latter is termed re-authentication. An EIS sign-on typically involves one or more of the following steps:

Determining a security principal under whose security context a physical connection to an EIS will be established.

Authentication of a security principal if it is not already authenticated.

Establishing a secure association between the application server and the EIS. This enables additional security mechanisms (example, data confidentiality and integrity) to be applied to communication between the two entities.

Access control to EIS resources

The Connector Architecture supports single sign on across multiple EISs. Single sign-on capabilities are useful in applications that need access to resources in multiple EIS systems. For example, an employee self-service application can give employees access to HR and Payroll records with a single sign on.

The Security Contract extends the Connection Management contract to support EIS sign re-authentication of pooled connections as necessary.

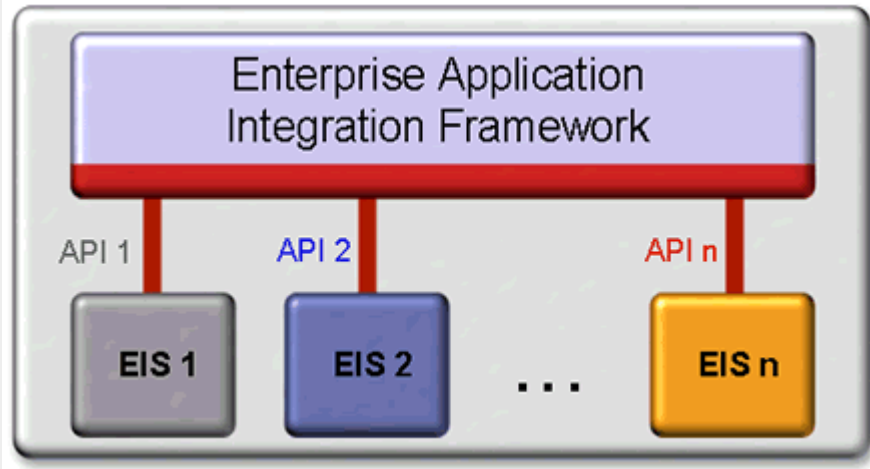
Common Client Interface

The CCI defines a standard client API for application components. The CCI enables application components and Enterprise Application Integration (EAI) frameworks to drive interactions across heterogeneous EISs using a common client API.

The target users of the CCI are enterprise tool vendors and EAI vendors. Application components themselves may also write to the API, but the CCI is a low-level API. The specification recommends that the CCI be the basis for richer functionality provided by the tool vendors, rather than being an application-level programming interface used by most application developers.

Challenges of Client Tool Integration

The heterogeneity challenges of EIS/application server integration also hold true for enterprise application development tools vendors and EAI frameworks. Typically, EISs provide proprietary client APIs. An application development tool or EAI framework needs to adapt these different client APIs to a higher abstracted layer. This abstracted layer raises the API to a common level on which tools and EAI vendors build useful functionality.



The CCI solves this problem by providing an API that is common across heterogeneous EISs. This avoids the need for tool and EAI vendors to adapt diverse EIS-specific client APIs. These vendors can use the CCI to build higher-level functionality over the underlying EISs.

Common Client Interface

The CCI defines a remote function-call interface that focuses on executing functions on an EIS and retrieving the results. The CCI is independent of a specific EIS; for example: data types specific to an EIS. However, the CCI is capable of being driven by EIS-specific metadata from a repository.

The CCI enables applications to create and manage connections to an EIS, execute an interaction, and manage data records as input, output or return values. The CCI is designed to be toolable, leveraging the JavaBeans architecture and Java Collection framework.

The 1.0 version of the Connector Architecture recommends that a resource adapter support CCI as its client API, while it requires that the resource adapter implement the system contracts. A resource adapter may choose to have a client API different from CCI, such as the client API based on the JDBC API.

JDBC API and Connectors

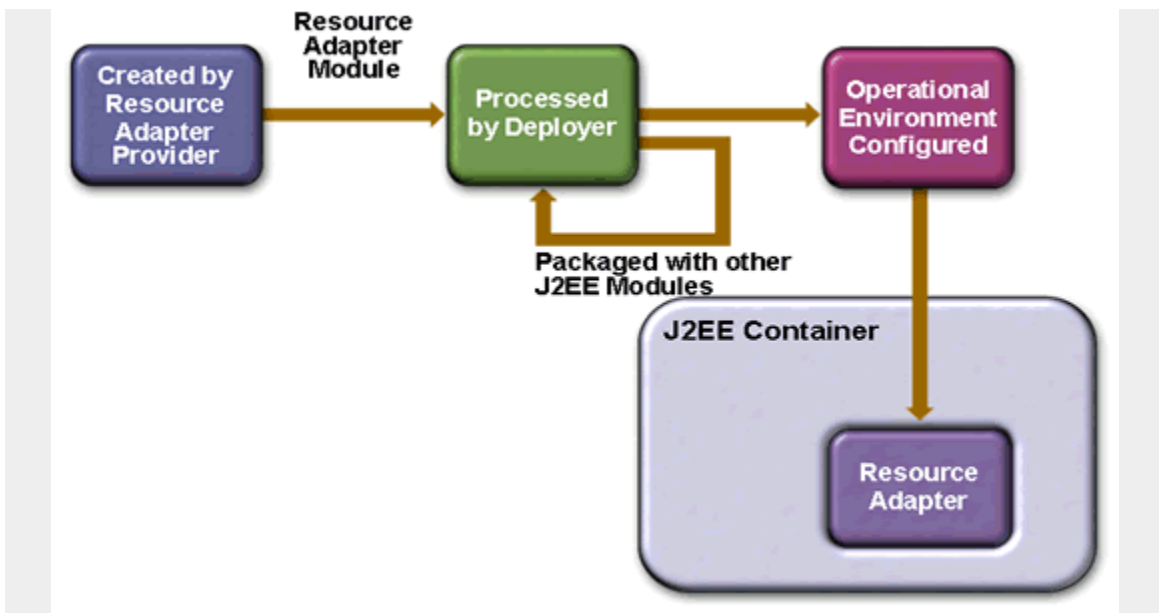
The relationship between the JDBC API and Connectors should be understood from the perspectives of application contract and system contracts.

The JDBC API defines a standard client API for accessing relational databases, while the CCI defines an EIS-independent client API for EISs that are not relational databases. The JDBC API is the recommended API for accessing relational databases while CCI is the recommended client API for other types of EISs.

At the system contract level, the connector SPIs may be viewed as a generalization and enhancement of JDBC 2.0 contracts. Future JDBC specifications may align with the Connector SPIs by offering it as an option with JDBC 2.0 SPIs. Another option for application server vendors is to wrap JDBC drivers under the Connector system contracts.

Packaging and Deployment

The Connector Architecture provides packaging and deployment interfaces, so that various resources adapters can easily plug into compliant J2EE application servers in a modular manner.



A resource adapter provider develops a set of Java interfaces and classes as part of its implementation of a resource adapter. These Java classes implement Connector Architecture-specified contracts and EIS-specific functionality provided by the resource adapter. The development of a resource adapter can also require use of native libraries specific to the underlying EIS.

The Java interfaces and classes are packaged together (with required native libraries, help files, documentation, and other resources) with a deployment descriptor to create a Resource Adapter Module. A deployment descriptor defines the contract between a resource adapter provider and a deployer for the deployment of a resource adapter.

A resource adapter module may be deployed as a shared, stand-alone module or packaged as part of a J2EE application. During deployment, the deployer installs a resource adapter module on an application server and then configures it into the target operational environment. The configuration of a resource adapter is based on the properties defined in the deployment descriptor as part of the resource adapter module.

Connector Architecture and Enterprise Application Integration

To illustrate the potential benefits of the Connector Architecture, this section provides two scenarios from different perspectives.

Integrating an EIS with Multiple Tools and Servers

A software vendor provides an ERP system focused on mid-sized manufacturing companies. Its customers are starting to build multi-tier, Java applications and want to build tightly coupled integration between these applications and the vendor's ERP system.

Although the ERP vendor may publish an API, not all of the application server vendors support it. Also the ERP system's customers are using a number of different application servers, presenting a potential logistical problem for the ERP vendor.

Instead of building or certifying interfaces for each system, the ERP vendor creates a single resource adapter using the Connector Architecture. This resource adapter implements Connector Architecture specified connection, security, and transaction contracts. The vendor then makes this adapter available, and informs customers that they can work with any compliant J2EE application server. The ERP vendor also implements the CCI in its resource adapter, opening up access directly to client components, or to a wide range of application development or EAI tools.

Using the Connector Architecture significantly reduces the ERP vendor's development efforts, giving it immediate integration and consistent operations with a wide variety of compliant J2EE tools and application servers.

Business-to-Business Commerce Solution

The following scenario illustrates the use of the Connector Architecture in a Business-to-Business supply chain solution.

Wombat Corporation is a manufacturer implementing a B2B e-commerce solution that improves interactions with its suppliers. Like most manufacturers, Wombat has enormous investments in its existing EIS systems, including an ERP system and a mainframe transaction processing system.

Wombat buys a compliant J2EE application server (called B2B server in this example) that supports interactions with multiple buyers/suppliers using XML and HTTP/HTTPS. Wombat integrates access to its EIS systems using off-the-shelf resource adapters that plug into the B2B server. Wombat can deploy as many resource adapters as it has EISs to integrate. An application server can "plug in" multiple resource adapters for the systems required for an application.

This scenario illustrates an important point: the Connector Architecture is designed for creating tightly coupled integration -- typically integration within the enterprise. Operations between different companies, such as a manufacturer and its supplier, are generally loosely coupled; for this, XML messaging is more appropriate.

The Evolution of the Connector Architecture

The Connector Architecture is a product of the Java Community Process program, an open process used by the Java community to develop and revise Java technology and specifications. Sun's partners in the Connector effort are EIS vendors, development tool vendors, and EAI vendors. Key participants include BEA, Fujitsu, IBM, Inprise, iPlanet, Motorola, Oracle, SAP, Sybase, Tibco, and Unisys. To date the standard experiences strong industry support, as all stakeholders stand to benefit from its adoption.

The 1.0 release of the J2EE Connector Architecture specification does not address all of the interfaces and system contracts that could potentially be required. The goal of the 1.0 release was to address the most pressing needs in a way that would speed industry adoption of the standard. For example, the 1.0 release specifies three system-level contracts, described above. These are mandatory components of the interface. Future system level contracts may address thread management and messaging through Java Message Service (JMS).

The Common Client Interface (CCI) is optional in the 1.0 implementation. The Connector Architecture does not address the issue of meta data for data representation and type mapping, which will certainly be relevant in the use of CCI. This issue will be addressed in future versions.

Summary

Just as the JDBC API extended the Java platform to integrate relational databases, the Connector Architecture extends the J2EE platform to integrate and extend the EISs that manage valuable processes and data in the enterprise. The Connector Architecture enables scalable, simplified access to valuable enterprise resources, without compromising data integrity or security on the EISs.

From: <http://java.sun.com/javase/overview/whitepapers/connector.jsp>