PERSISTENT

Java SAP R/3 Integration

November 2004

Abstract

Enterprises of all descriptions have long sought to integrate and manage all their information assets within a single system. The goal remains elusive because enterprises create and store their assets in a myriad of disparate systems - relational databases, mainframes, different operating systems, hierarchical repositories, and so on. Using new and existing assets in an efficient, integrated, and interchangeable manner has become the key to surviving, and thriving in the new economy.

SAP remains the world leader in enterprise-software domain, with over 11000 installations worldwide. With over 60% of market share in ERP segment, it becomes an obvious choice for the EAI and EII product vendors to target SAP R/3 Enterprise as a data source, and therein emerges the need for integration. With the emergence of Java as a platform for developing enterprise applications and its portability across different Operating Systems, it becomes the obvious platform for integration with SAP R/3 Enterprise.

Keywords: SAP R/3, RFC, BAPI, IDoc, JCo

Author: Manoj Gangwani

Persistent Systems Private Limited

PERSISTENT WHITE PAPER

Contact details

Persistent Systems Private Limited,

"Bhageerath", Senapati Bapat Marg,

Pune, India 411 016
Telephone: +91 (20) 2567 8900
Fax: +91 (20) 2567 8901
Email: whitepapers@persistent.co.in
Web: http://www.persistent.co.in

Publication and copyright details

TABLE OF CONTENTS

Introduction

Aim

This section provides you an overview of the SAP R/3 Enterprise, different integration interfaces exposed by it and integration of an application with SAP R/3 using Java platform. This white paper assumes that you already know Java. Some knowledge of SAP is also desirable, but you should be able to get by even if you don't know much about SAP. This document will cover all aspects of integration with SAP R/3 Enterprise using Java as a platform.

Intended Audience

This document is intended for developer(s)/architect(s) embarking on EAI/EII projects involving integration with SAP R/3. They will find this document suitable for obtaining a quick overview of the SAP R/3 Enterprise and different integration interfaces exposed by the same. It will also give them a quick overview of the SAP's Java Connector, with some details on the issues involved in such integration projects.

SAP R/3 Enterprise

Overview

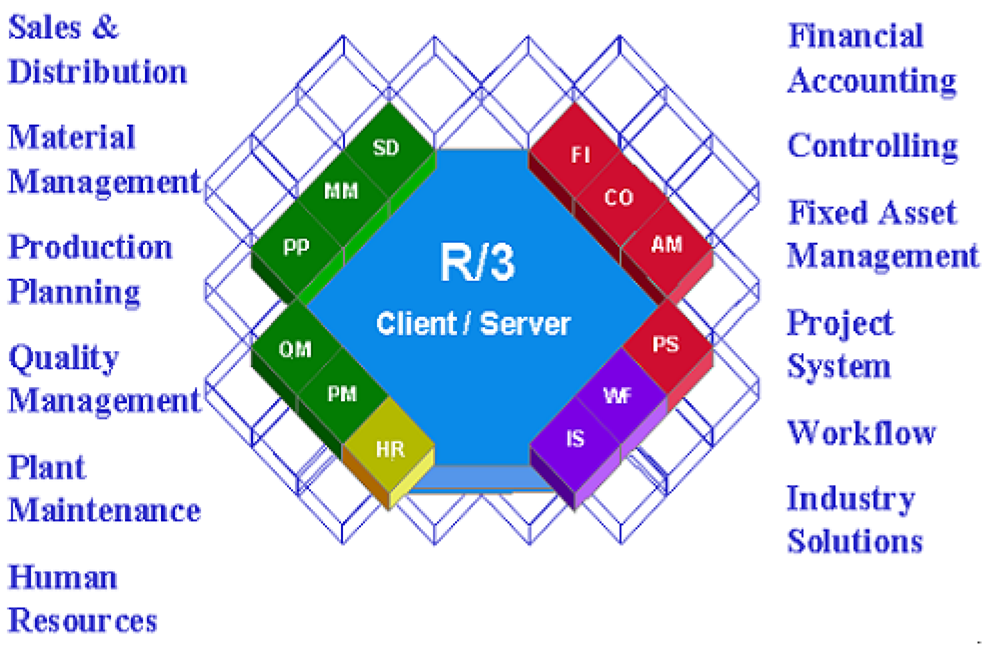
SAP's R/3 System has set new norms for standard software that can be universally implemented. R/3 uses advanced development techniques to achieve comprehensive integration of business administration and data processing. It combines state-of-the-art technology with comprehensive business administration functionality to provide a fully integrated business solution for an enterprise.

SAP stands for Systems Applications and Products for data processing.

Architecture

SAP R/3 is event-driven transaction processing software for business events in an organization's primary value chain. Transaction processing systems supported by enterprise software are most concerned with the day-to-day needs of a business in conducting its on-going activities.

Figure 1: SAP R/3 Enterprise Architecture



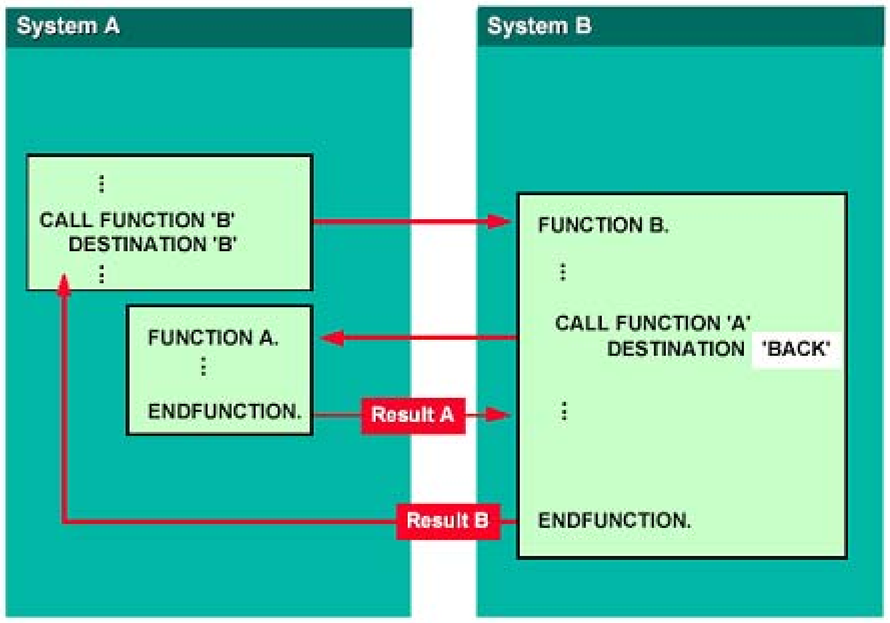
The R/3 Basis is the software that implements SAP three-tier client/server architecture. It consists of application modules and application servers, which are distinct components. The application modules support all of a company's business transactions and are integrated interactively. All application modules share data through the R/3 database, which contains the data for all modules.

Integration Interfaces

Remote Function Calls (RFC)

A Remote Function Call (RFC) is the call of function module in a partner system. The caller is the RFC client and the called partner is the RFC server. RFC is based on the known RPC model from the UNIX-TCP/IP environment. RFC in SAP environment is based on a CPI-C interface implemented by SAP.

Figure 2: Remote Function Call



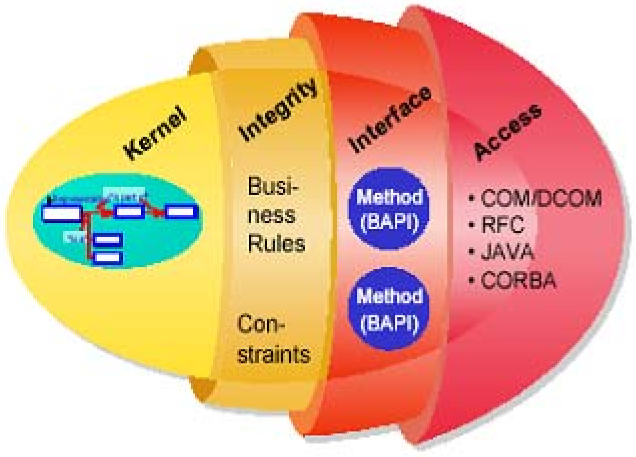
The remote call of a function module is easier for an application programmer to use than program-to-program communication, because data is exchanged only using predefined parameters.

SAP BO, BOR and BAPI

SAP Business Objects (BO)

Business model objects for the real world (for example, employees, customer orders, and so on) are reproduced in R/3 as business objects. A business object encapsulates the business processes and the data linked to it. These objects can be seen as being multi-layered.

Figure 3: SAP Business Object (BO)



The innermost layer is made up of the actual data and its structures. The second layer maintains integrity. It represents the business logic of the object and describes the business rules and limitations for the business object. The interface layer describes the implementation and structure of the object. The interface layer allows access to the attributes, methods, and events that are defined for the object.

SAP Business Object Repository (BOR)

The Business Object Repository (BOR) contains the SAP business object types and SAP interface types as well as their components, such as methods, attributes, and events.

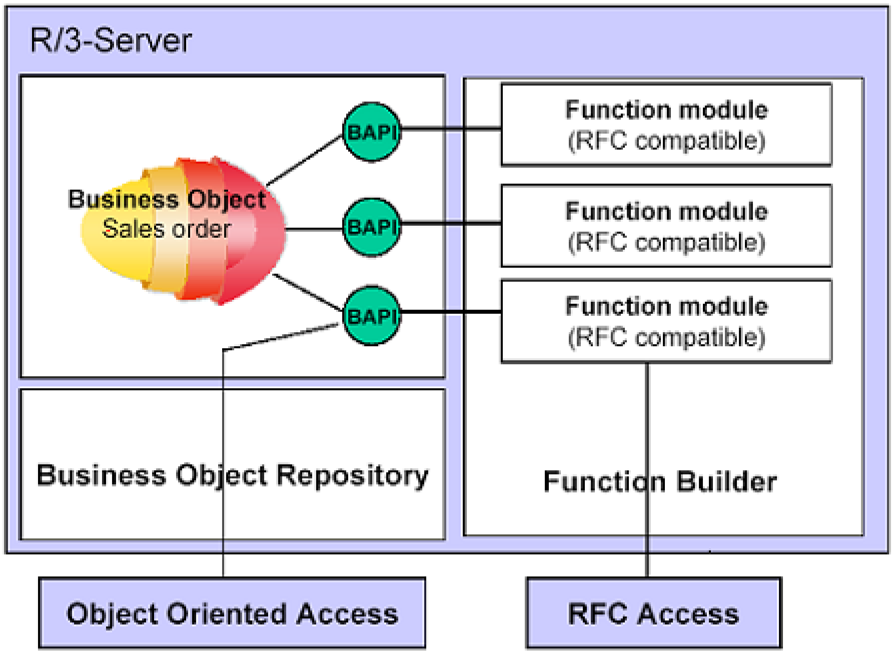
The BOR has the following functions:

- Enables an object-oriented view of the R/3 System data and processes.
- Arranges the various interfaces in accordance with the component hierarchy.
- Manages BAPIs in release updates.

SAP Business Application Programming Interface (BAPI)

SAP BAPIs provide developers with stable, reliable interfaces for accessing the functionality offered by SAP business objects. BAPIs can reduce the development cost by providing built-in functionality through predefined methods of the SAP business objects. These methods can reduce programming time in deployment projects.

Figure 4: SAP R/3 BOR and BAPI

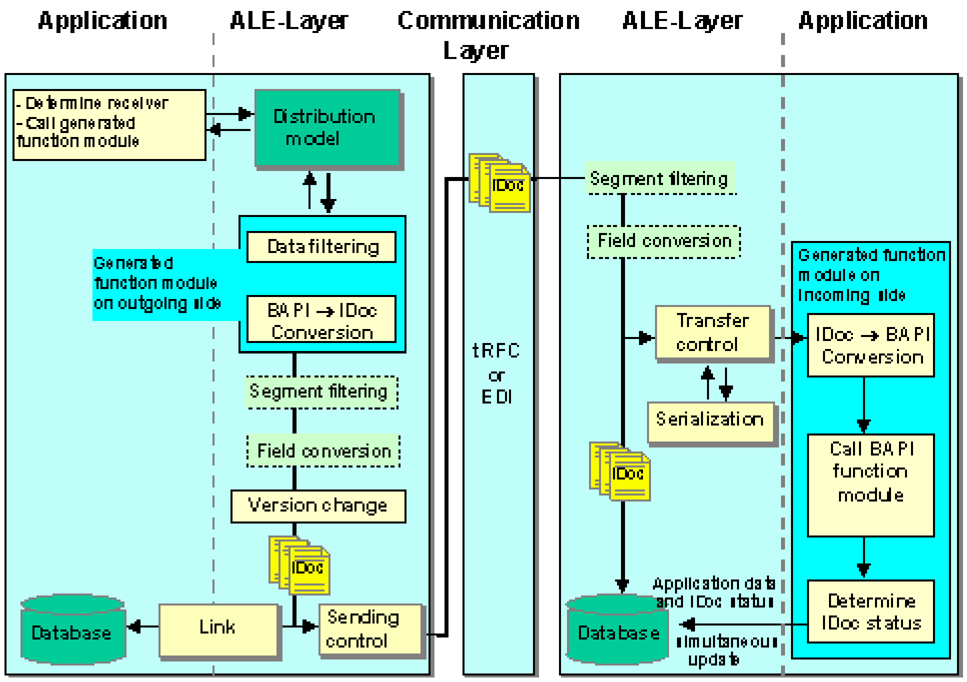

R/3-Server
Business Object
Sales order
BAPI
BAPI
BAPI
Function module (RFC compatible)
Function module (RFC compatible)
Function module (RFC compatible)
Business Object Repository
Function Builder
Object Oriented Access
RFC Access

Application Link Enabling (ALE) and IDocs

Application Link Enabling (ALE) is an important middleware tool in SAP's Business Framework Architecture. ALE is the exchange of messages controlled from a business point of view. It can integrate business processes between R/3 systems and non-R/3 systems as well as between R/3 systems.

Application systems are loosely coupled in an ALE integrated system. Data is exchanged asynchronously, whereby the data arrives in the receiver system, even if the receiver system cannot be connected to at the time the data is sent. ALE uses synchronous communication for receiving data only.

Figure 5: Application Link Enabling (ALE)



An Intermediate Document (IDoc) is a container for exchanging data between R/3, R/2 and non-SAP systems. Non SAP-systems can use IDocs as the standard interface for data transfer. The IDoc interface consists of the definition of a data structure and the processing logic for this data structure.

Java SAP R/3 Integration

SAP Java Connector (JCo)

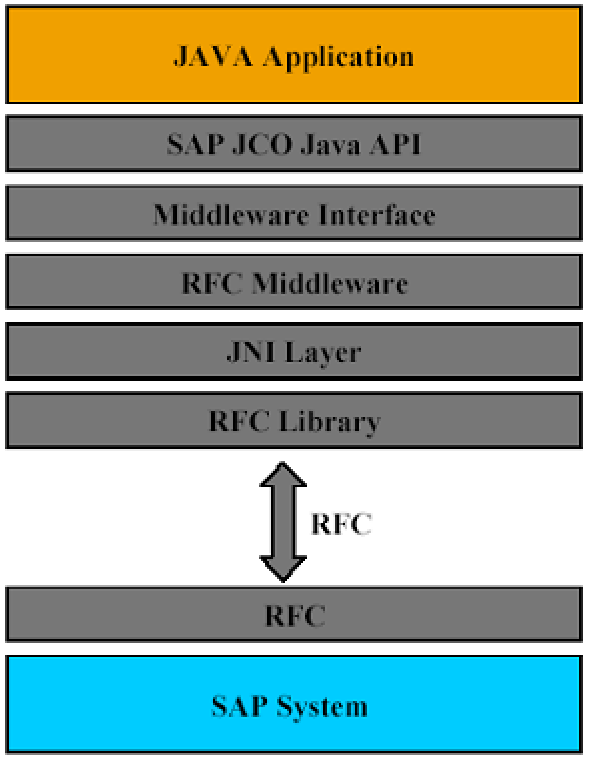
SAP's java middleware, the SAP Java Connector (JCo) allows SAP customers and partners to easily build SAP-enabled components and applications in Java. It enables communication between SAP System and Java (supports communication with SAP R/2, R/3 3.1 and later).

Some salient features of JCo are as follows:

- Supports both inbound (Java calls ABAP) and outbound (ABAP calls Java) calls.
- Supports Unicode and Non-Unicode systems.
- Hides all difficult parts like codepages, data type conversions, connection pooling, and so on from the programmer.
- Hides RFC details from the programmer.

- Hides DDIC details from programmer.
- Consistent and easy-to-learn class design and API.
- Dynamic metadata lookup and caching.

Figure 6: SAP Java Connector (JCo)



The java application uses only JCo's java apis for connecting with SAP R/3 Enterprise. RFC middleware uses RFC Library through JNI layer for connecting to the SAP R/3.

Implementing JCo Client

Creating Connection Pool

A connection pool is created by calling method addClientPool(). [st1]

Creating Repository

A Repository object is created by calling method createRepository(). A Repository object contains meta-data for all function modules at runtime. [st2]

Creating Function Template

A function template object is created by calling method getFunctionTemplate(). [st3]

Creating Function

A function object is created by calling method getFunction(). [st4]

Getting Connection

A connection object is retrieved from connection pool by calling method getClient(). [st5]

Providing Inputs

Inputs are passed to the function module by calling method getImportParameterList(). [st6]

Executing Function

A function module is executed by calling method execute(). [st7]

Accessing Output

Output of the function module is accessed by calling method getExportParameterList(). [st8]

Destroying Connection Pool

A connection pool is destroyed calling method removeClientPool(). [st9]

Issues

Performance

Connection Pooling

Connection pools are critical for the performance of SAP R/3 as well as for application integration due to the following:

- Overhead of logging on to SAP system is avoided because the connection stays open and can be reused.
- Maximum number of connections concurrently used, are restricted thus preventing the use of too many SAP resources.

However, care should be taken not to create performance bottleneck in an application, by making the maximum number of connections too small. The SAP system must be configured so that it can accommodate extra load created by application.

Use a Fresh Function Object

Efforts to optimize performance of an application by reusing existing JCO.Function objects, is not only superfluous but also dangerous because JCO.Repository buffers metadata for function modules. If a function module is called in SAP that fills a table parameter without deleting existing rows, incorrect results will occur since more and more rows are added to the table.

This must be avoided by using a fresh JCO.Function.

Use Only One Repository

Creating one repository object per user of the application, or, even worse, for each session of the user, significantly increases number of calls to the SAP system. This will not only degrade performance of the application, but also the SAP system.

It can be avoided by using JCO.Repository class. JCO.Repository object dynamically retrieves and caches metadata (read-only) information for function modules from SAP. The function module metadata in the repository is client-independent; so one repository per SAP system is sufficient. This can be accomplished by writing a suitable repository manager class.

Inactive Table Parameters

Some function modules, especially BAPIs, cover a lot of application scope and thus need to have quite a few table parameters. Applications might only need a subset of the supported tables. The performance of an application can be improved by inactivating those table parameters that an application does not utilize. This can be accomplished by invoking the setActive() method, available for both the JCO.ParamaterList and JCO.Request object types.

Appending Multiple Table Rows

While appending multiples row to a JCO.Table parameter, application will run faster if calls to the appendRow() method are replaced by one call to appendRows(int num_rows).

Synchronization

In order to optimize performance, JCo itself synchronizes access to JCO.Pool and JCO.Repository objects. Everything else is not synchronized. In a multi-threaded environment, care should be taken while sharing objects (like JCO.Table objects) between different threads. Sharing connections (JCO.Client objects) is disallowed and will lead to an exception.

BAPI, State, and Commit

Most function modules (including most BAPIs) are stateless. SAP does not remember anything between calls in the same session. The connection must be released back to the pool after finishing one uninterrupted activity in the application, if all the function modules invoked are stateless.

Most updating BAPIs require an additional external commit call to actually cause any change on the SAP database to happen. This allows us to combine multiple update BAPI calls into one LUW. The commit call must happen in the same SAP session in which we called the update BAPI(s). In other words, the connection needs to be preserved till the time end of the LUW is reached.

Appendix

Terms

Business Framework Architecture (BFA): Business Framework Architecture is a component-based architecture enabling software components from SAP and other software vendors to communicate and be integrated with each other.

Advanced Business Application Programming (ABAP): All application programs in SAP business applications are created in Advanced Business Application Programming (ABAP), SAP's own, interpretive language.

Function Module (FM): Function Modules are the ABAP programs deployed in the SAP R/3 system for implementing additional program logic.

Function Builder: Function Builder is a SAP tool that can be used to generate, test, and administer function modules in a SAP R/3 system.

CPI-C: Common Programming Interface Communication a platform-independent API that interfaces to a common set of APPC (Advanced Program-to-Program Communication) verbs.

JNI: Java Native Interface is the native programming interface for Java that is part of the JDK.

EAI: Enterprise Application Integration is unrestricted sharing of data and business processes throughout the networked applications or data sources in an organization.

EII: Enterprise Information Integration provides seamless integration of disparate data sources on an enterprise scale, provides strategic advantage organizations require.

RPC: Remote Procedure Call is calling a procedure that need not exist in the same address space as the calling procedure.

LUW: Logical Unit of Work is a set of transactions where either all are successfully applied against the database, or none have any impact on the database.

Sample Code

Shown below is the code snippet to understand how java can be used for invoking function module in SAP R/3 Enterprise. The code snippet explains inbound (Java calls ABAP); by invoking BAPI BAPI_SALESORDER_GETLIST for fetching a list of sales order from SAP R/3 Enterprise.

```
import com.sap.mw.jco.*;

public class SalesOrder {

        static final String SID = "R3";

         // The repository we will be using
```

```
IRepository repository;
public SalesOrder() {
          try {
                    // Add a connection pool to the specified system
                    JCO.addClientPool( SID,        // Alias for this pool
                    10,         // Max. number of connections
                    "000",      // SAP client
                    "johndoe",  // userid
                    "*****",    // password
                    "EN",       // language
                    "appserver", // host name
                    "00" );

                    // Create a new repository
                    repository = JCO.createRepository("PSPL", SID);
          }
          catch (JCO.Exception ex) {
                    System.out.println("Caught an exception: \n" + ex);
          }
}

// Retrieves and displays a sales order list
public void salesOrders(){
          JCO.Client client = null;
          try {
                    // Get a function template from the repository
                    IFunctionTemplate ftemplate = repository.getFunctionTemplate(
                    "BAPI_SALESORDER_GETLIST");

                    // if the function definition was found in backend system
                    if(ftemplate != null) {
                              // Create a function from the template
                              JCO.Function function = ftemplate.getFunction();

                              // Get a client from the pool
                              client = JCO.getClient(SID);

                              // Fill in input parameters
                              JCO.ParameterList input = function.getImportParameterList();
```

```
input.setValue("0000001200", "CUSTOMER_NUMBER"   );
input.setValue(      "1000", "SALES_ORGANIZATION");
input.setValue(         "0", "TRANSACTION_GROUP" );
```

```
// Call the remote system
client.execute(function);
```

```
                // Print return message
                JCO.Structure ret =
                function.getExportParameterList().getStructure("RETURN");
                System.out.println("BAPI_SALES_ORDER_GETLIST RETURN: "
                + ret.getString("MESSAGE"));
```

```
// Get table containing the orders
JCO.Table sales_orders =
function.getTableParameterList().getTable("SALES_ORDERS")
;
```

```
                // Print results
                if (sales_orders.getNumRows() > 0) {
                        // Loop over all rows
                        do {
                                System.out.println("-----------------------");
                                // Loop over all columns in the current row
                                for (JCO.FieldIterator
                                e=sales_orders.fields();
                                e.hasMoreElements(); ) {
                                        JCO.Field field = e.nextField();
                                        System.out.println(field.getName()
                                        + ":\t" + field.getString());
                                }//for
                        } while(sales_orders.nextRow());
                }else {
                        System.out.println("No results found");
                }//if
        } else {
                System.out.println("Function not found in R/3 system.");
        }//if
}
catch (Exception ex) {
        System.out.println("Caught an exception: \n" + ex);
}
```

```
                finally {
                        // Release the client to the pool
                        JCO.releaseClient(client);
                }
        }
        protected void cleanUp() {
                // Destroying connection pool
                JCO.removeClientPool(SID);
        }
        public static void main(String[] argv){
                SalesOrder order = new SalesOrder();
                order.salesOrders();
                order.cleanUp();
        }
}
```

Reference

Books

[1] Enterprise Java for SAP (Indian Edition) by Austin Sincock

Internet

[2] Tutorial (JCo Tutorial.pdf) shipped with JCo software

[3] Help available at SAP Help Portal http://help.sap.com

[4] Help available at http://ifr.sap/com

[5] Help available at http://sapgenie.com/