Interoperability of Java-Based Applications and SAP's Business Framework State of the Art and Desirable Developments

Markus Aleksy, Axel Korthaus
University of Mannheim, Germany
{aleksy|korthaus}@wifo3.uni-mannheim.de

Abstract

As the leading vendor of enterprise business standard software, SAP has recognized the need to adapt their R/3 system to current trends in software development and to meet market needs for speed of development, flexibility, openness and interoperability. In this paper, we first present SAP's approach to object-oriented and component-based technology by describing the Business Framework, the concepts of Business Objects, BAPIs, and the Business Object Repository. On this basis, we then analyze current communication architectures and products enabling the interaction of external Java-based software applications with SAP R/3, point out the advantages and disadvantages of different solutions and finally elaborate on potential strategies and steps for driving the evolution of SAP R/3 in order to further increase interoperability, openness and flexibility.

1. Introduction

The integrated business software system SAP R/3 covers a wide range of business software requirements from accounting and controlling through human resources management to project management. Being SAP's vital software product, R/3 has to be reworked continuously in order to meet the changing market needs and to face competition. Recognizing the demands for interoperability, flexibility, openness, and the trends in software development being based on object-oriented and component-based paradigms, SAP has designed a new architectural framework for R/3 which has made the relatively monolithic system open up and become an integration platform for using so-called "business objects" across business processes and applications (cf. [14]). This component- and business object-oriented architecture is called the Business Framework [12].

In this paper, we present the concepts of this approach and focus especially on the possibilities for integrating R/3 with third party applications. Our emphasis lies on Java-based programs that are meant to interact with R/3. We critically analyze current products that support Java-based access to R/3 and we suggest an evolution strategy for R/3 to further improve interoperability, openness and flexibility.

2. SAP's Business Framework

The SAP Business Framework [12] is a software architecture that organizes the R/3 functionality into a structure of business components and a business object model. By using this component- and object-oriented approach, the Business Framework significantly contributes to a reduction of complexity and thus facilitates the dealing with and the understanding of the comprehensive R/3 system. In a way, the Business Framework can be seen as an object-oriented interface to or a view of the R/3 system, offering many of the advantages of an object-oriented approach while the benefits of the non object-oriented but "mature" R/3 core system are maintained.

While a high level of system integration has been maintained in the Business Framework, it has become possible to recombine business components such as "Product Data Management" and "Consolidation and Pricing" originating from different R/3 releases. Furthermore, integration with third party components (e.g., produced by customers or partners) has become much easier.

Figure 1 shows the basic architecture of the Business Framework. The Business Framework consists of:

- *SAP Business Components:* SAP Business Components represent a self-contained software application domain in the enterprise. Each business component consists of a number of SAP Business Objects, whose interfaces, the BAPIs, jointly constitute the interface of the component. Business processes can be implemented entirely within one business component, or they may cross several components.

Because of the encapsulation of functionality and the accessibility through clearly defined and stable interfaces, each SAP Business Component has its own software lifecycle, e.g., each component can be maintained independently of other components.

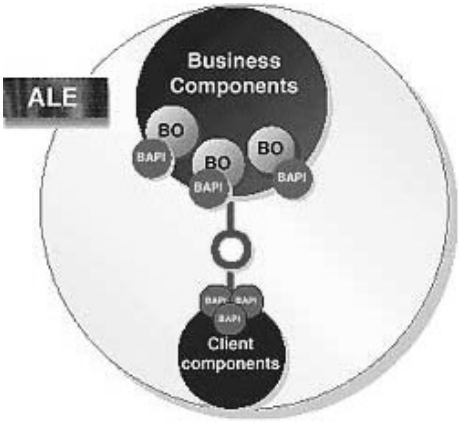

Figure 1: Business Framework ([12], p. 6)

- *SAP Business Objects and Business Application Programming Interfaces (BAPIs):* SAP Business Objects are units of data and functionality with clearly defined business semantics, i.e., they represent a business concept from the real world. Following object-oriented principles, business objects encapsulate business data and functionality (i.e., SAP Transactions). Each business object shows a certain "behavior" which is specified by its interfaces, the so-called Business Application Programming Interfaces (BAPIs). The BAPIs determine the external appearance of a business object, i.e., they are what can be seen and used by an external software program (another business object, business component, or an external application program). In order to utilize a business object's services, these BAPIs have to be invoked.
- *Communication and integration technologies:* In order to enable the access to BAPIs on a broad basis, communication services such as Microsoft's DCOM, SAP's RFC etc. are supported by the framework. The integration of business objects distributed across system boundaries, e.g., in order to execute business processes that involve multiple R/3 and third party systems, can be achieved by using an integration service called Application Link Enabling (ALE).

Before we will elaborate on the aspects concerning communication technologies and the access layer of SAP Business Objects (cf. Figure 2) in Section 3, the following subsections will give an overview of SAP's object and component model, laying the foundation for the considerations in the remainder of this paper.

2.1. SAP Business Objects

SAP Business Objects [13] are abstractions of real world objects from the business domain, such as "Employee", "Product", or "Customer", encapsulating implementation details of data and functionality. With the help of business objects, the developer's view of R/3 is significantly simplified, not only because business objects map to real world business concepts, but also because they shield the developer from having to deal with thousands of R/3 data entities and ABAP/4 transactions. Thus, SAP Business Objects not only provide a high-level business-oriented view suitable for modeling and business engineering purposes, but also represent a programming interface to the R/3 system (cf. [5]).

The concept of SAP Business Objects shows the core characteristics of an object-oriented approach: business objects as instances of business object types, encapsulation of data and functionality, inheritance between business object types, and polymorphism in hierarchical inheritance relationships. A business object is characterized by its identity, its attributes and methods, and the events it produces or consumes. Identification of a business object is achieved with the help of SAP Business Object Identifiers that consist of the SAP System Identifier, the name of the object type, and a key. SAP Business Objects are structured into the following layers (cf. Figure 2):

- *business object kernel*: contains object data and the core business logic,
- *integrity layer*: contains business rules and constraints needed to manipulate the object data in a consistent way,
- *interface layer*: defines the services provided by the object, the input event control and the output events published by the object,
- *access layer*: describes technologies (COM/DCOM, Java, CORBA) that can be used to access the object.

Access to a SAP Business Object and its data can only be gained by calling its public methods. An application program that wants to make use of the services of a business object needs to know how to call these methods (i.e., it needs information about the name of the business object, the name of the method and information about input and output parameters), but does not have to know anything about the implementation lying behind those methods. The interfaces of a SAP Business Object which

present the object's services (its public methods) are realized by BAPIs.

Figure 2: Multiple layers of a SAP Business Object ([13], p. 7)

2.2. Business Application Programming Interfaces (BAPIs)

Business Application Programming Interfaces (cf. [9]-[11], [13]) provide the information needed to be able to access SAP Business Objects, i.e., to call their methods. SAP tries to establish BAPIs as a global communication standard for business applications and is committed to keeping the BAPIs stable. Thus, the modification of parts of the system (e.g., of a business component) will not affect other parts of the system, as long as the dependency between the parts is restricted to contracts based on BAPIs which remain unchanged. For example, investments in customer-specific applications that rely on BAPIs are preserved, even if the customer migrates to a new R/3 release.

SAP Business Objects and their BAPIs are stored in the Business Object Repository (BOR) which is described in the following subsection. Currently, the implementation of the business object methods is based on SAP Function Modules, which have to be called using the SAP-specific protocol Remote Function Call (RFC). So, although there is an object-oriented abstraction layer for viewing and accessing R/3 functionality in the form of SAP Business Objects and their BAPIs now, the underlying implementation is not object-oriented at all. The mapping of BAPIs to function modules that implement them is also stored within the BOR. In order to be able to access SAP Business Objects, i.e., to invoke their BAPIs, a connection to an R/3 server instance and an access authorization are needed, and information about the name of the business object, the BAPI and its input/output parameters must be available.

There are basically two ways of accessing business objects, either in a purely object-oriented manner by calling a BAPI directly via the BOR, usually with the help of proxy classes in an object-oriented language, or by making a conventional remote function call (RFC) aimed at a lower level function module implementing the BAPI.

2.3. Business Object Repository (BOR)

As part of the R/3 Repository, the Business Object Repository [1] serves as R/3's object-oriented information management store which contains all the SAP Business Objects and their methods, presented as BAPIs, in addition to technical objects (such as text, notes, etc.) and metaobjects (for the description of business object types, attributes, methods, events etc.). As the central access point for external applications, the BOR has two main functions:

- The BOR is used for the identification and description of the business object types and their BAPIs and it contains all the information needed to integrate business object type definitions and BAPI or RFC calls into an application. The attributes, methods, and events available from an object can, for example, be retrieved from its object type, which is documented by a metaobject in the BOR. Furthermore, the BOR provides the function modules needed for the implementation of the BAPIs.
- The BOR is the runtime environment in which instances of business objects are created on requests from client applications. Thus, the client does not have to be aware of the coding and location of a business object.

To allow the access to SAP Business Objects whose interfaces are not known at compile time, the BOR provides a Dynamic Invocation Interface.

3. Java-based access to SAP R/3

Having presented the basic concepts of SAP's Business Framework approach, we will now concentrate on questions regarding the interoperability provided by this approach. Since Java [16] is one of the most interesting and successful modern object-oriented programming languages, we will emphasize Java-based communication architectures and products for the interaction with R/3.

3.1. Characteristics of different communication architectures

Choosing the "right" communication architecture is not trivial and plays an important role with regard to

- the integration of different programming languages, operating systems, and computer architectures,
- the integration of existing applications (compatibility),
- portability of source code,
- performance, and
- scalability.

For a Java-based application, there are four possible technologies on which communication with the SAP R/3 System can be based:

- Sun's Remote Method Invocation (RMI) [18],
- Sun's Java Native Interface (JNI) [17],
- Microsoft's Distributed Component Object Model (DCOM) [6],
- OMG's Common Object Request Broker Architecture (CORBA) [7].

Since version 1.1, Sun's RMI has been shipped together with the Java Development Kit (JDK) and therefore has been made available to all users of a JDK-compliant development environment. The disadvantage of RMI is that it only supports Java-to-Java invocations. Direct connections to applications not written in Java (such as the R/3 system) are impossible.

The second alternative, JNI, is also a component of the JDK (and therefore also shows the advantage of general availability) and enables accessibility between Java and other programming languages. A considerable disadvantage of this alternative is that the non-Java code might not be transferable to other platforms (e.g., in the case of assembler) so that one of the main advantages of Java is invalidated: its portability. Being connected with the latter, another weakness of JNI as the basis of a middleware technology is its lacking suitability for use in applets.

The DCOM technology, which has been developed by Microsoft, on the one hand does allow the integration of different programming languages, but, on the other hand, is mainly provided for Windows platforms. Another disadvantage is the fact that in spite of a licensing agreement with Sun Microsystems, Microsoft does not supply a 100% compatible Java version, so that source code portability will be restricted to Windows platforms if Microsoft-specific Java features like accesses to ActiveX are used.

The last alternative is CORBA, which combines most of the advantages of the alternatives mentioned above, such as the independence of the programming language, the operating system and the computer architecture, and furthermore avoids their general disadvantages. On the other hand, the complexity of the CORBA specification is considerable and developers have to acquire detailed knowledge in order to be able to build CORBA-based solutions. A CORBA-compliant runtime library is already part of Java 2 (formerly known as Java 1.2). As soon as an IDL-to-Java compiler is also included, no more additional software will be needed for the development of distributed Java applications, unless the developer is dependent on the use of certain services, such as an event, security, or licensing service, or the use of an interface and/or implementation repository.

3.2. Available software products for Java-based access to SAP R/3

For some time now, a number of products have been offered which enable the access of Java-based applications to SAP's R/3 System. Among them are:

- Visual Edge's ObjectBridge and Madrid for R/3 ([19], [20]),
- IBM's Visual Age for Java Enterprise [4] and
- SAP's Java Remote Function Call (JRFC) [14].

Moreover, there are several libraries for other programming languages (such as C, C++, Delphi, or Perl) which contain programming constructs for the communication with the R/3 system. In the following, we focus on the presentation of the Java-based products.

ObjectBridge and Madrid for R/3 CORBA

We mention Visual Edge's products ObjectBridge and Madrid for R/3 only for the reason of completeness. They have not been part of our analysis, since they have not been made available to us despite of several requests.

Madrid is the successor to ObjectBridge, about which we do not possess any detailed information. Madrid does not contain a middleware architecture of its own, but is designed to collaborate with different architectures such as CORBA, (D)COM, or RFC. As described in [20], it is shipped together with several adapters for COM, CORBA (Orbix, OrbixWeb, VisiBroker, PowerBroker, ObjectBroker, as well as a generic IIOP adapter), Enterprise JavaBeans (EJB), and Tuxedo. The central component of Madrid is the Business Object Browser. With the help of this browser, business objects can be seen from different views (e.g., the ABAP or CORBA view).

For the reason of providing independence of the underlying middleware alone, Madrid seems to be very interesting. However, as noted above no detailed analysis has been possible for us.

Visual Age for Java Enterprise—Access Builder for SAP R/3

Visual Age for Java Enterprise is a development environment that allows access to the R/3 system with the help of the SAP Access Builder for SAP R/3. While there is a version 1.1 available in the meantime, in our examination we have analyzed version 1.0 of the product. Communication is possible with R/3 system versions 3.1g through 3.1i and 4.0b. Version 4.0a is not officially supported due to problems with the R/3 metainformation, although access is possible in many cases, too. Communication with the R/3 system is executed via RFC calls, which are implemented using JNI (which implies the disadvantages mentioned in subsection 3.1). IBM's decision to employ JNI instead of CORBA is surprising, because IBM is not only known to produce very heterogeneous hard- and software (e.g., Component Broker—a CORBA-compliant object request broker), but also to be an active member of the Object Management Group (OMG) which has specified the CORBA standard. However, a support for connections based on the Internet Inter-ORB Protocol (IIOP) is already planned for the near future.

The development environment itself supplies a prefabricated JavaBean (a Logon-Bean), the BOR-Browser for accessing metainformation that is stored in the BOR, and interfaces to SAP's BAPIs. The proxy objects generated from the BOR information are compliant with Sun Microsystems' JavaBeans specification [2]. The conversion between big-endian and little-endian representation schemes is done automatically. This also holds true for JRFC, which is described below. A big disadvantage of this solution is, however, that country-specific characters like the German umlauts are converted incorrectly, which again also is the case for JRFC. A positive aspect of both solutions is the extensive source code portability, which can be attributed to the fact that IBM and SAP have jointly developed the classes in the Java packages **`com.sap.rfc`** and **`com.sap.rfc.exceptions`**.

Further information about Access Builder can be drawn from the experience report [3].

SAP's Java Remote Function Call (JRFC) package

For a certain period of time now, SAP has been providing the JRFC package which allows access to the R/3 system from Java via RFC. From the beginning of 1999 on, JRFC version 1.0 has been part of the product SAP Automation 4.5 and is no longer shipped as a stand-alone product. JRFC is compliant with the Java Development Kit 1.1 and supports multithreading on the client side as well as on the server side.

A great advantage of SAP's solution compared with IBM's product is the possibility for developing applets which are able to communicate with the R/3 system. At the moment, the package is only available for Windows 95/98/NT, and, in addition to the SAP JRFC classes, it contains parts of IONA's object request brokers (ORBs) Orbix 2.3c (C++) and OrbixWeb 3.0 (Java), through which parts of the communication are passed. The RFC calls produced by a Java application are delivered to the JRFC server via IIOP [7] and transformed into SAP-specific RFC calls by the server. Figure 3 illustrates the communication process.

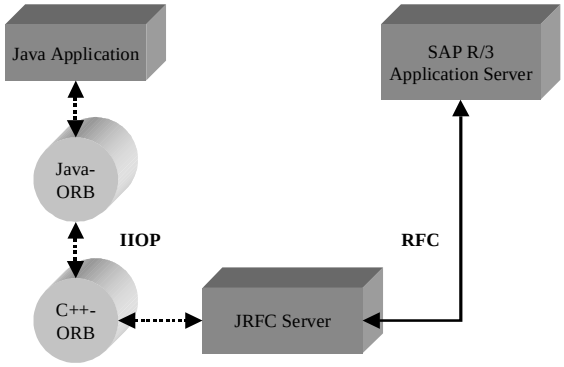

Figure 3: Access to R/3 via JRFC

Another component of the Automation package is a tool called SAP Assistant. It serves as a BOR and RFC browser and allows the creation of Java proxy objects for the invocation of BAPIs. The base class from which all Java proxies of SAP Business Objects are derived is contained in the package **`com.sap.bapi`** and is called **`jboBase`**. For example, code generation for a business object **`ProfitCenter`** which has a method **`Getlist`** creates the proxy class source code files

`jboProfitCenter.java`,
`jboProfitCenterGetlistParams.java`

and several source code files of classes needed for the access to table parameters, table rows and structural parameters. In our example, these are:

`jboBapi0015_1TableRow.java`,
`jboBapi0015_1Table.java`,
`jboBapi0002_3Structure.java`,
`jboBapi0015_2Structure.java`, and
`jboBapireturnStructure.java`.

Another important class in the package **com.sap.bapi**, besides **jboBase**, is **jboKey** which serves for the identification of single instances of business objects and with the help of which key fields can be read or determined.

4. Differences between IBM's and SAP's solutions

In this chapter, fundamental differences between the two solutions that we have examined in detail are analyzed with respect to the aspect of source code portability.

As already mentioned before, the Java classes contained in both products originate from a joint effort by IBM and SAP. The goal pursued in this effort was the achievement of extensive source code portability to facilitate the exchange of programs between Access Builder and JRFC. However, we came across several differences during our analysis, although it is mostly a question of some minor deviations, such as the differing number of constructors of the types **SimpleInfo**, **UserInfo** or **SystemInfo** or differences regarding single methods such as the method **setGlobalConnection** of the class **IRfcConnection**, which is present in the IBM product but is missing in the SAP solution. On the whole, several classes are affected by structural differences, mostly concerning additional methods or attributes in one of the two products.

Substantial differences can be found in the **Exception** classes (from package **com.sap.rfc.exception**) and in the class **MiddlewareInfo** (from package **com.sap.rfc**), which stores information concerning the kind of middleware in use.

The JRFC package contains 25 different exception classes whose names start with **JRfc** in each case and which are declared as subclasses of **JRfcBaseException** or **JRfcRuntimeException**, respectively. IBM's solution comprises nearly twice as many classes as SAP's product, including all the exception classes found in JRFC except for **JRfcRemoteServerException**.

Using IBM's Access Builder to compile **Client.java**, an example program delivered together with SAP's JRFC package which has a size of 8 KB and can be found in the directory **RfcGetCustomer**, no less than 15 errors do occur. 11 of those 15 errors are caused by differences in exception handling, i.e., exceptions which are thrown remain uncaught or handlers are declared to catch exceptions which simply cannot occur. With respect to source code portability, this aspect is definitely in need of improvement.

The type of middleware chosen not only influences the potential of developing applets serving as R/3 clients, but also inevitably affects the structure of the class **MiddlewareInfo** which serves for storing corresponding information.

In the process of establishing a connection to SAP R/3 from SAP's JRFC or IBM's Access Builder, there are minor differences concerning the source code that has to be written. For example, during the initialization phase, the **MiddlewareType** has to be set for the **MiddlewareInfo** object. Using JRFC, the attribute has to be set to the value **middlewareTypeOrbix**, whereas the value must be **middlewareTypeJNI** in the case of Access Builder.

Table 1 contains a summary of the differences between the **MiddlewareInfo** classes provided by the two solutions. In the first two rows of the table, those exceptions and attributes of IBM's and SAP's **MiddlewareInfo** class are arranged in pairs that have the same semantics but are named differently. The third row shows a number of methods, which can be found in SAP's JRFC package, but are not present in IBM's Access Builder.

Table 1 : Differences concerning the MiddlewareInfo class

	IBM	SAP
Exceptions:	**IllegalArgumentException** **NullPointerException**	**JRfcIllegalArgumentException** **JRfcNullPointerException**
Attributes:	**middlewareTypeJRFC** **middlewareTypeStringJRFC**	**middlewareTypeOrbix** **middlewareTypeStringOrbix**
Methods:		**clone()** **getApplet()** **getLocalDomainName()** **getOrbDiagnostics()** **setApplet()** **setLocalDomainName()** **setOrbDiagnostics()** **toString()**

5. Critical review and potential improvements

A considerable disadvantage of JRFC and its components is their platform dependence. The complete package is only provided for Windows

95/98/NT (Intel) and, therefore, developers are inevitably tied to these platforms. For example, the software component implementing the JRFC server is an "exe"-file which can only be executed under Windows. If the developer wishes to use ORBlets (Java applets that employ CORBA as communication middleware), he will be forced to run the JRFC server and the web server on the same Windows computer which might lead to the need of restructuring the existing enterprise information technology infrastructure, since the accommodation of the web server is often the domain of a Unix-based machine.

A similar criticism applies to the SAP Assistant component. Just like the JRFC server, SAP Assistant is only delivered as an executable Windows binary. Taking into account that BAPI accesses can only be made using Java proxy classes generated by the SAP Assistant tool, Windows is again the obligatory platform.

In summary, it may be said that—in spite of the platform independence of CORBA and the portability and variety of platforms supported by Java—choosing the JRFC approach means being restricted to Windows platforms and not being able to exploit all the possibilities offered by the CORBA and Java technologies.

On the other hand, during our analysis it turned out that the client side in the JRFC solution is portable to other platforms. In contrast to IBM's Access Builder, SAP's solution is not based on an operating system specific runtime library, but uses the CORBA-compliant product OrbixWeb. The Java class files needed by a JRFC client can thus be separated into two groups, namely

- OrbixWeb runtime library classes and
- SAP JRFC client classes.

The OrbixWeb runtime library classes (`IE.Iona`...) are the implementations of the classes specified in the CORBA standard (`org.omg`...). The JRFC client classes have been developed by SAP and use the OrbixWeb classes.

In order to investigate the platform independence of both groups of classes, we began by transferring the SAP JRFC client classes from Windows to Solaris and tested them in the context of an OrbixWeb (version 3.0) installation which was already in use on the Solaris workstation. After this experiment, we migrated both the SAP JRFC classes and the OrbixWeb runtime library classes to a Linux platform. As a result, we found out that the example programs included in the package as well as our own example programs could be compiled and executed without any problems.

5.1. Suggestions for improvement

Although there are some first signs of interoperability and openness, the drawbacks stated before remain. In order to address these problems, one can think of a number of potential steps to be taken for driving the evolution of R/3. The following suggestions for improvements are not only directed to SAP as the manufacturer of R/3 itself, but also aimed at the great number of SAP customers and users who should jointly articulate their demands for increasing the openness of R/3 and its possibilities for integration with various platforms and programming languages.

We structure our suggestions for improvement according to the problem scope they address. The first proposition limits itself to measures concerning the JRFC server, whereas the second suggestion takes a more comprehensive view and focuses on improvements of the accessibility of the business objects themselves.

Publication of IDL interfaces for the JRFC server

The first idea of improving openness and flexibility would be to enhance the existing JRFC approach in the following way. SAP should consider

- the publication of the JRFC server's IDL interfaces, together with
- the publication of the JRFC server's initial reference,

which are two aspects that might be of a certain interest to users who mainly need to access the R/3 system via CORBA, irrespective of the platform on which the client should run.

Being based on the object request broker Orbix, JRFC is already CORBA-based, but until now SAP has been refraining from publishing the—obviously existing—IDL interfaces. A solution in favor of publishing the JRFC server's IDL interfaces could not eliminate the dependence on the Windows platform completely (since the execution of the JRFC server and the generation of proxy classes by SAP Assistant would still have to be performed on a Windows machine), but it would enable CORBA-based access from any platform for which a CORBA-compliant ORB is available, because the structure of the JRFC server becomes visible through the IDL interfaces and the server location becomes known by interpreting the initial reference. Prerequiste of the combination of different ORBs is their general ability to interoperate not only in theory, but also in practice. In a study in-

volving several ORBs of different vendors, we were able to show that this prerequisite is met [15].

Although this solution would guarantee the possibility for platform independent access to R/3 from Java and even allow different programming languages to access the JRFC server via its IDL interfaces, it would still have the major disadvantages mentioned before.

The first of these is the remaining problem of platform restriction concerning the JRFC server, i.e., a Windows platform is still required on which the server can be run, which implies the already indicated consequences regarding the web server. On the other hand, having the server IDL interfaces at hand, even completely new but compatible versions of an RFC server in any language for which an IDL mapping has been defined could be implemented.

A second but similar inconvenience is the fact that the generation of Java proxy classes would still have to be performed by the Windows-based SAP Assistant tool which is not available for different platforms yet. As a consequence, the classes needed for BAPI invocation and generated under Windows would have to be transferred to the desired platform, for example by FTP. When using other programming languages than Java, the developer has to accept the fact that only pure RFC calls are possible, since there is no BAPI proxy generation available for programming languages such as COBOL, Smalltalk, or Ada.

Publication of IDL interfaces for BAPIs

The second idea goes further than just involving the JRFC server. The central aspects of our second suggestion are

- the publication of IDL interfaces for BAPIs and
- the publication of the initial reference of a corresponding business object server, which conforms to the IDL interfaces and can be developed by SAP itself or by any third party vendor.

With this approach, the developer would be released from his dependence on the Windows operating system, since the use of the JRFC server and the SAP Assistant component would no longer be required. Thus, accesses from programming languages which were not supported before would be enabled, and, furthermore, additional operating systems and hardware platforms could be integrated.

Table 2 contains a summary of the pros and cons of the different approaches stated above. Since the BAPI/IDL approach seems to be the most beneficial, it will be the focus of attention in the following subsection of our paper.

Table 2: Overview of the benefits offered by the suggested approaches

Approach / Criterion	State of the art (JRFC)	JRFC-IDL	BAPI-IDL
Platform independent access to R/3 from Java	✓	✓	✓
Access to R/3 from other programming languages than Java via RFC	—	✓	✓
Independence of the Windows operating system as the obligatory platform	—	—	✓
Direct access to SAP-BAPIs from other programming languages than Java	—	—	✓

5.2. Prospects of publishing IDL interfaces for SAP Business Objects

Since SAP is responsible for the definition of the BAPIs, it should also be the authority responsible for the publication of the IDL interfaces, although it might be advantageous to cooperate with vendor-independent consortia such as the OMG, which might result in a general and binding standard. This could not only guarantee stable IDL interfaces (as long as the BAPIs themselves remain stable), but would also lead to the possibility of combining or exchanging different products based on these interface standards. Choosing an IDL-based approach, the JRFC server would not be needed any more (or would be transformed to a business object server on the server side, respectively) and real platform independence could be achieved (cf. Figure 4).

In addition, an IDL-based approach would have further advantages. For example, access to business objects would not be restricted to Java, but could also be gained by other programming languages such as COBOL, Ada or Smalltalk. Formerly unsupported operating systems and hardware platforms could be integrated. Existing applications which communicate with R/3 via RFC or (D)COM would not have to be modified because of this ap-

proach, since the R/3 basis would remain unchanged. Consideration of existing legacy applications is an absolute necessity, e.g., for the reason of investment protection.

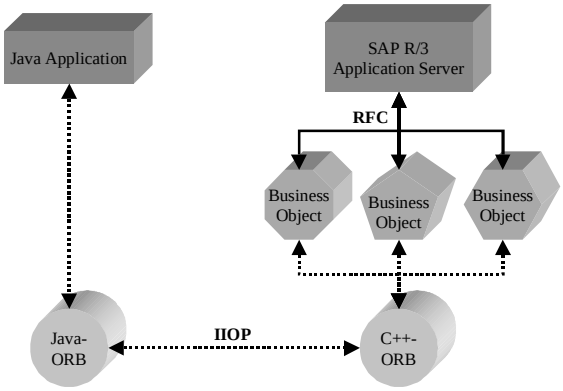

Figure 4: Direct connection to R/3 via CORBA

In the approach suggested here, the CORBA-based business objects must still communicate with R/3 on the basis of RFC, but the various CORBA-based clients would gain free access to the business objects.

A more comprehensive approach could be the integration of an ORB in the core R/3 system. However, this would only be profitable to CORBA-based applications, e.g., with respect to the direct communication via IIOP. As mentioned before, dispensing with RFC support is impossible because of the need for compatibility. Interoperability with (D)COM could be provided by existing packages such as SAP Automation / DCOM Connector or by a (D)COM-CORBA bridge. Nevertheless, the use of different middleware architectures could be problematic. Furthermore, this solution would require a considerable amount of change to the core SAP R/3 System.

In an idealized vision of the future, R/3 might evolve from a relatively monolithic system to a framework of transaction-based business object components.

Irrespective of which of these potential evolution paths might become reality, the clients' view of R/3 is only influenced by the definition of IDL interfaces, so that no effects on the client implementation have to be expected. The clients call methods of objects. Whether these objects do the work completely by themselves or whether they simply delegate the requests is not important for the clients.

The publication of IDL interfaces would enable other vendors to develop their own business object servers for the access to R/3. The objects in an interoperable solution would have to be registered on a naming service [8], and the service's initial reference and the names of the objects would have to be published.

Which effects different ways of implementing business object servers would have is the subject of the following section.

5.3. Design considerations for business object servers

Today, the only realistic way of encapsulating R/3 by business object servers which can be accessed via CORBA/IIOP is to design one monolithic business object server module whose only task is to map the incoming BAPI calls to RFC calls directed to the R/3 core system. This approach would be the easiest and fastest solution to achieve CORBA-based business object accessibility.

On the other hand, this kind of design would not reflect what an object technology purist would expect from a "good" business object server: ideally, each object should show specific behavior and provide information corresponding to its real world counterpart in a way as self-contained as possible. This would, of course, imply that parts of the data structures and functionality of R/3 would have to be duplicated in (in the case of leaving R/3 mainly unchanged) or migrated to (in the case of breaking up the R/3 system) the business object server(s).

Duplication of data elements and functionality, however, would lead to many problems concerning network traffic and data consistency. Because of the redundancy, corresponding data stored in the business object server(s) and in R/3 would have to be aligned very frequently, namely every time an access by a RFC client (directly to R/3) or a CORBA client (to a business object server) occurs. Because of these problems and the poor architectural design, this solution is out of the question in practice.

The most interesting alternative would be a completely new design for the R/3 system, distributing its data and functionality across intelligent, self-contained business object servers that do not need any monolithic system behind them.

In this vision of the future, R/3 functionality would be provided by a framework of distributed, concurrent, transaction-based business object components with stable interfaces, which could be easily recombined, exchanged, and integrated with each other and with components from different vendors or customer-specific components in a plug-and-play scenario and on a platform-independent basis. The realization of this vision would require the replacement of the existing R/3 core system by newly designed business object servers. Although the definition of BAPIs is paving the way for the migration to real business object components, it seems to be unrealistic, with respect to the immense

effort and the existing customer base involved, to expect such a development in the foreseeable future.

Breaking up the R/3 system into business object servers does not necessarily need to result in a one-to-one mapping of current SAP Business Objects to business object servers and therefore leads to a number of questions concerning architectural design considerations. At least the following possibilities for the implementation of the business objects have to be taken into account :

- one server per business object,
- one server per business component (e.g., Controlling, Finance, etc.),
- one server for each collection of objects that communicate with each other very frequently,
- one server for all business objects.

Each of these design approaches has its pros and cons, and a discussion of which approach would be the most promising goes beyond the scope of this paper.

5.4. Determination of the initial reference

Our suggestion for improving interoperability and openness by publishing IDL interfaces corresponding to BAPIs consisted of two parts: first, the publication of the IDL interfaces themselves and second, the publication of the initial reference of a business object server. The aspect of how to get the initial server reference might at first appear to be trivial, but it plays an important role if the CORBA connection has to be vendor-independent. Up to now, the Object Management Group (OMG) has left it to the software vendors to implement the way how a client may obtain the initial Inter-ORB Reference (IOR) of a remote server.

However, two different ways have been defined by the OMG [7], but they both have some disadvantages. The first possibility is the use of the method **`ORB.resolve_initial_references()`**, which provides an initial reference, e.g., of the naming service. However, this approach is vendor-specific, since the OMG has not described how the ORB itself can obtain this reference. Usually, this is done by handing over command line parameters, producing configuration files or setting up environment variables. For that reason, this approach will have to be avoided if the application under development has to be portable.

The second possibility is the use of the ORB methods **`object_to_string()`** and **`string_to_object()`**. These methods serve for the conversion of an object reference into a string or the generation of an object reference from a string, respectively. Although this procedure is vendor-neutral, it does not answer the question how the client can obtain the object reference.

Common ways to solve this problem are:

- look up in a file which is provided on the client as well as on the server (e.g., via NFS or AFS)
- transmission across the network (e.g., via FTP or email)

Not only does this kind of relationship between the client and the server run counter to the CORBA idea of location transparency, it furthermore represents a potential source of errors. Since it is possible that a certain amount of time passes by between retrieval and use of the IOR, it might occur that the IOR has become invalid in the meantime. This aspect is especially relevant in the context of applets, because an applet requires the IOR to be hard-coded into the corresponding HTML page. As a compromise, a program/script can be developed which retrieves the IOR and inserts it into the HTML page.

ORBs such as VisiBroker or Orbix/OrbixWeb provide solutions (“agents” in VisiBroker and “demons” in Orbix/OrbixWeb) which are more comfortable, but do not conform to the standard. With these products, the initial reference of the server can be obtained by calling the method **`bind()`** which is not part of the CORBA standard. As already stated before, the JRFC package is based on the ORBs Orbix/OrbixWeb, so that the invocation of the **`bind()`** method is exactly the way how the initial reference of the JRFC server has to be found out. However, if ORBs from other vendors are to be used, different ways of determining the initial IOR will be required.

For SAP, the easiest solution to this problem would be to use a naming service on which all the business objects are registered, and to describe

- the names registered for the objects and
- the location where the initial IOR of the naming service is stored.

6. Conclusion and outlook

After some years of paying lip-service to Java/CORBA without doing much to prove its support, SAP has been engaging strongly in this area recently. The JRFC package strongly indicates SAP’s increasing commitment to Java and CORBA. In this paper, we have suggested to consider the publication of IDL interfaces for the JRFC server and especially for SAP Business Objects. In order to get on with and to intensify its

Java/CORBA commitment, SAP should see this approach as a good chance to meet customer demands for flexibility, openness and interoperability and to preserve or even enhance its competitiveness.

However, until this approach will be taken, users who want their applications to interface with SAP R/3 depend on the available solutions, such as IBM's Access Builder for SAP R/3 and SAP's JRFC package. We have shown that both the IBM solution and the SAP solution can be further improved. Some of the deficiencies described in our paper regarding IBM's product are said to be remedied in the already available version 1.1 (source code portability) and in the upcoming version 1.2 (IIOP support). SAP, on the other hand, has achieved big improvements in version 1.0 of JRFC compared with the beta version. For example, the documentation has been amended significantly and has become much more detailed. Furthermore, not only (J)RFC calls are supported now, but also BAPI invocations are possible. Nevertheless, SAP has still missed to provide a vendor-independent solution. Therefore, most of SAP's customers are still forced to use Windows as their platform.

The publication of IDL interfaces for SAP Business Objects and of the initial reference needed for communication could solve this problem and allow interoperability of R/3 and formerly unsupported platforms and programming languages such as Ada, COBOL or Smalltalk. Furthermore, existing solutions for access to R/3 would have to face up to an increased competition, since the publication of IDL interfaces corresponding to BAPIs and the description of a way to obtain an initial reference would lead to a growing market of CORBA-based solutions provided by different vendors.

References

[1] Benchmarking Partners (1996): "The SAP R/3 Business Object Repository"; in: Strategic Technologies, 1 (1) 1996

[2] Hamilton, G., editor (1997): "JavaBeans API Specification"; ver. 1.01, Sun Microsystems Inc., July 24, 1997, http://www.javasoft.com/beans/docs/beans.101.pdf

[3] Herrmann, J., Reipa D. (1999): „Der Monolith öffnet sich: eine objektorientierte SAP R/3-Schnittstelle für Java"; in: OBJEKTspektrum, 2/99, pp. 63-69

[4] IBM (1999): General IBM web site containing information about Visual Age for Java and Access Builder for SAP R/3, http://www.software.ibm.com/ad/vajava/sap.htm

[5] Korthaus, A. (1997): "Business Objects as Constituents of Future Distributed Business Information Systems"; discussion paper 1/97, University of Mannheim, 1997, ftp://ftp.wifo.uni-mannheim.de/pub/PEOPLE/korthaus/CBOpaper.ps

[6] Microsoft Inc. (1999): "Distributed Component Object Model (DCOM)"; General Microsoft web site containing links to information about the DCOM Technology, http://www.microsoft.com/com/dcom.asp

[7] OMG (1998): "CORBA/IIOP 2.2 Specification"; OMG Technical Document Number 98-07-01, http://www.omg.org/corba/corbaiiop.html

[8] OMG (1997): "Naming Service Specification"; OMG Technical Document Number 97-12-10, ftp://www.omg.org/pubs/docs/format/97-12-10.pdf

[9] SAP (1997): "BAPI Introduction and Overview"; SAP white paper, Version R/3 Release 4.0, Dec. 1997, http://www.sap.com/products/techno/bapis/edu/docu/caalbe/caalbe.htm

[10] SAP (1997): "BAPI Programming"; SAP white paper, Version R/3 Release 4.0, Dec. 1997, http://www.sap.de/products/techno/bapis/edu/docu/caalce.doc

[11] SAP (1999): "Open BAPI Network"; General SAP information web site containing information about BAPIs, http://www.sap.de/bapi

[12] SAP (1997): "R/3 System—Benefits of the Business Framework"; SAP white paper, Aug. 1997, http://www.sap-ag.de:80/products/techno/pdf/50016302.pdf

[13] SAP (1997): "R/3 System—SAP Business Objects"; SAP white paper, Aug. 1997, http://www.sap-ag.de:80/products/techno/pdf/50016301.pdf

[14] SAP (1999): "SAP Business Technology—Enabling Solutions for your Business"; General SAP web site containing links to information about SAP's Business Technology approach, http://www. sap.de/bfa/

[15] Schader M., Aleksy M., Tapper C. (1998): „Interoperabilität verschiedener Object Request Broker nach CORBA2.0-Standard"; in: OBJEKTspektrum, 3/98, pp. 72-77, http://www.wifo. uni-mannheim.de/ IIOP

[16] Sun Microsystems Inc. (1998): "Java Platform 1.2 API Specification"; JDK 1.2, 1998, http://java.sun.com/products/jdk/1.2/docs/api/index.html

[17] Sun Microsystems Inc. (1997): "Java Native Interface Specification"; JDK 1.1, May 16, 1997, http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html

[18] Sun Microsystems Inc. (1998): "Java Remote Method Invocation Specification"; Revision 1.50, JDK 1.2, Oct. 1998, http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html

[19] Visual Edge (1999): "Madrid for R/3"; General Visual Edge web site containing information about Madrid for R/3, http://www.software.ibm.com/ad/vajava/sap.htm

[20] von Arb, R. (1999): "Madrid: Cross-Application Process Integration for SAP"; in: SAP Technical Journal, 1 (1), 1999, http://www.saptechjournal.com/1999.01/review/index.html