High Noon: Why ABAP Performs Better in Portal Development than Java

By Axel Angeli, logosworld.com and Lynton Grice, Consultant

Editor's Note: *Axel Angeli and Lynton Grice face off in this defining showdown of Java versus ABAP as they assess which is the most effective development environment for SAP Portals applications. Pull off your boots and get comfortable while Axel and Lynton show you the good, the bad, and the ugly aspects of each environment in this comprehensive, 35-page discourse for developers. The authors give a detailed account of the effort put forth in implementing a portal application in Java, and then two years later, a "nearly identical" application in BSP. Axel and Lynton use side-by-side code examples and feature comparisons to demonstrate significant issues such as finding the right technical skills, language complexity and learning curves, database needs, and transport and version management for each language. The goal? To identify the best environment to support infrastructure stability and production agility By the end of the paper, you'll know why these authors think that ABAP BSP means "high noon" for Java.*

Introduction

This article describes the experiences of implementing a nearly identical SAP Portal application, first in Java, and two years later, in BSP. This article attempts to give a practical viewpoint on Java and BSP development in the Portal production environment. It has not been written to promote Java over BSP or vice versa, but rather to distil some "ground level" insight into how the languages compare in complexity and "user friendliness" from a development perspective—while still being able to get the job done!

The Right Strategic Choice for Portal Development

The strategic choice a company makes with regard to the development environment in the enterprise can have a huge impact on that company's agility and productivity going forward. Whatever choice the company makes, it will require a serious amount of investment to establish an infrastructure for production, testing, and development, along with permanent costs in training (for the necessary technical skills). Through a series of coding examples and side-by-side comparisons, we will assess the pros and cons of both ABAP and Java development. By the end of the paper, readers will have a clear sense of why we feel ABAP and BSP remains the preferred option for most Portal development projects.

The "standards-based" approach of the SAP Portal has given companies the flexibility to make best use of their current programming skills, while still being able to produce the rich Portal content demanded by the business. Getting a programmer to learn a new programming language can be a very daunting task and often requires a deep paradigm shift in mindset. Take, for example, the task of trying to transform a traditional R/3 ABAP developer into a modern day Java developer—this is more often than not a VERY difficult move. Java, BSP, .NET, and WebDynpro are the matrix of Portal technologies that need to be carefully understood, when deciding on a strategic Portal development language.

Finding the Best Practical and Economical Solution

Sure, a company can "mix and match" here and there, but making best use of your current resources should be your first priority. Reading about these "Portal technologies" is one thing—understanding them on the practical level is another. It is not only the question of Java or ABAP, but also of the environment used. You must never forget that allowing several development frameworks will mean doubling the education for the maintenance team, sharing precious time (for gaining experience between contending areas), and dealing with possible conflicts in collaboration during production. On the other side, it should not be underestimated that a friendly competition between technologies can have an acceleration effect on productivity and innovation. ABAP Objects got the final push towards object orientation mainly due to the rivalry with Java. So the sum of TCO (Total Cost of Ownership) and ROI (Return on Investment) is the final value.

Getting an SAP Portal developer's viewpoint on development language choice is by no means an easy task. The Java developer will stress the openness and flexibility of the open source software over the proprietary approach delivered by SAP. The BSP developer, on the other hand, will probably talk about the SAP transport system, debugging features, and integrated repository to get his case across. Finding a fully "bi-lingual" SAP Portal developer in both Java and BSP/ABAP (with a good amount of business knowledge in both) is not easy.

This discussion tackles this problem head-on by portraying certain segments and findings by one such "bilingual" developer (certified in both Java and ABAP) over the past two or so years. As the authors come from the two different ends of technology, the respective benefits and disadvantages of the language framework could be clearly worked out. Here are the findings:[1]

J2EE Versus SAP Web AS

Although a separate discussion on its own[2], the known robustness and reliability of the underlying application framework is a very important one that influences application development directly. The SAP kernel has proved its reliability and stability over the years, and is definitely a force to be reckoned with when analyzing application frameworks. Although J2EE has numerous benefits to the extended enterprise, the relatively "young" framework still has numerous questions surrounding its efficiency and stability.

In our Java development experience for EP 5.0, a relatively simple code error had the potential to cause the J2EE engine to crash. Besides analyzing the accompanying Java "error stack trace" and accompanying log files, the error resolution could potentially also involve the "SAP Portal administrator". His or her role would typically include reviewing J2EE configuration files and/or applying one or more SAP notes (and possibly re-starting the J2EE engine). This did not help in speeding up the development cycle of Java iViews. On the positive side, however, the newer version of the SAP Portal (EP 6.0) seems to have improved in "robustness" over the earlier releases of EP 5.0, with regard to the J2EE engine, etc., and general user friendliness.

As mentioned earlier, the SAP application framework is known for its stability and reliability. In my BSP development experience for EP 5.0, I have never seen or even heard of a scenario (or piece of ABAP code) that caused the SAP instance to crash.

On the application framework level, SAP Web AS can definitely compete directly with J2EE; both in regard to development and in terms of general support for "open standards". In fact, probably one of the biggest advantages of the SAP Web AS framework is the seamless integration of the

[1] It should be noted that the authors come from the two different ends. While Lynton comes from the "Java world" into the "SAP world", Axel comes from the traditional ABAP line. Both have done considerable amounts of development in each language.

[2] See "The Clash of the Titans" in the SAPtips Document Library in the Web AS and NetWeaver™ section. i

development environment with the repository and data dictionary. J2EE cannot compete with SAP Web AS on this level.

Under the code name “Unbreakable Java”, SAP is currently putting in a great deal of effort in order to build a new J2EE framework on top of the reliable ABAP engine. Subtitled as the “Always On Java” initiative, it addresses and critiques the robustness of the J2EE implementations. But it will take some time until it is available to the public, so it is still worthwhile to concentrate on the current J2EE releases.

A Sample Portal Application

When trying to determine whether one programming language is easier to use than another, it is no use comparing them on the “Hello World” level. Some form of complex “nuts and bolts” code needs to be written and analyzed in order to have a true comparison.

Also, just repeating the alleged object-oriented features of a language does not help further. Completeness, flexibility, and readability are high value items to consider. The language should be able to map common design patterns one by one into appropriate code. In these points, both ABAP and Java perform only moderately well compared to the “ideal” programming language. But let us do some practical things to get a feeling of the essentials with respect to Portal development.

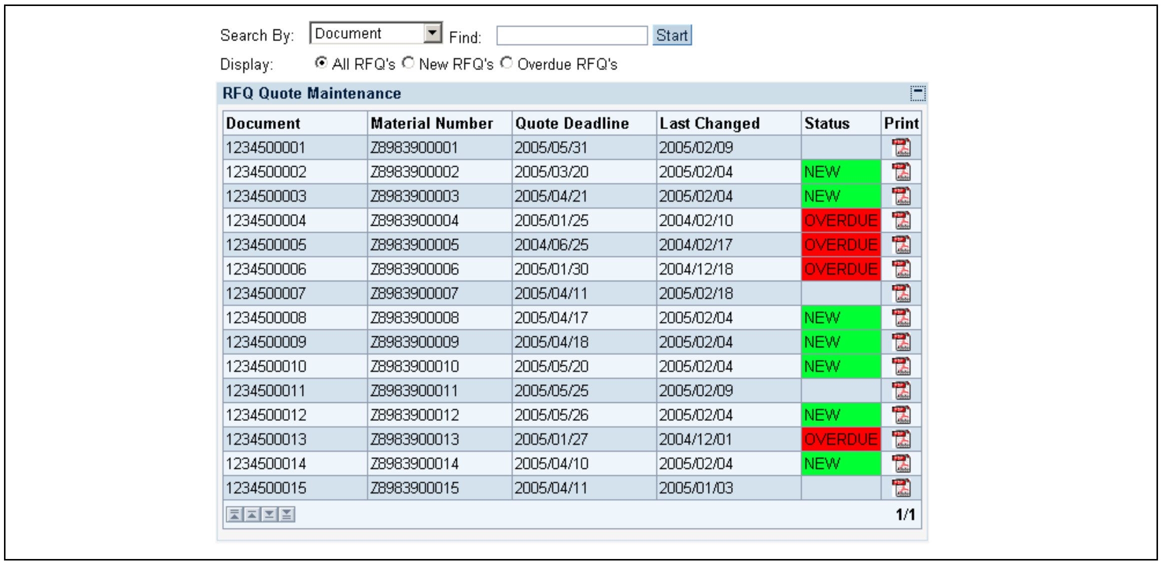
As an example, the solution we have chosen is to enter a Request for Quotation (RFQ) via the SAP Enterprise Portal that we implemented first in Java, then a good while later, again in BSP.

The RFQ procedure is a fairly complex process in SAP. Typically, transactions ME42 and ME43 are used to change and display RFQ data in SAP. Allowing a supplier to view, update, and print RFQ information directly from the SAP Portal has numerous time and cost saving benefits to a company.

Figures 1 and 2 show a particular “RFQ iView” that was first developed in Java and later in BSP. Some of the functionality provided in the iView includes:

- Printing the selected RFQ (via a custom SmartForm)
- Updating “pricing conditions” related to an RFQ
- Adding “tooling information” to an RFQ
- Viewing material, quality, engineering, transport. and packaging information for an RFQ

These two screenshots of the RFQ iView show the typical functionality provided.



Search By: Document Find: Start

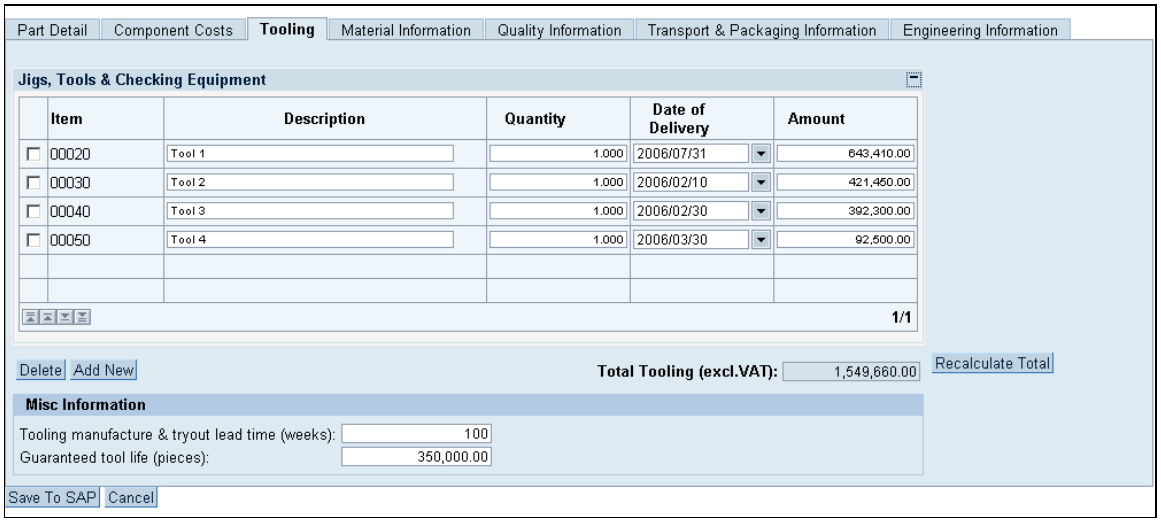
Display: [✓ All RFQ's] New RFQ's Overdue RFQ's

RFQ Quote Maintenance

Document	Material Number	Quote Deadline	Last Changed	Status	Print
1234500001	Z8983900001	2005/05/31	2005/02/09		
1234500002	Z8983900002	2005/03/20	2005/02/04	NEW	
1234500003	Z8983900003	2005/04/21	2005/02/04	NEW	
1234500004	Z8983900004	2005/01/25	2004/02/10	OVERDUE	
1234500005	Z8983900005	2004/06/25	2004/02/17	OVERDUE	
1234500006	Z8983900006	2005/01/30	2004/12/18	OVERDUE	
1234500007	Z8983900007	2005/04/11	2005/02/18		
1234500008	Z8983900008	2005/04/17	2005/02/04	NEW	
1234500009	Z8983900009	2005/04/18	2005/02/04	NEW	
1234500010	Z8983900010	2005/05/20	2005/02/04	NEW	
1234500011	Z8983900011	2005/05/25	2005/02/09		
1234500012	Z8983900012	2005/05/26	2005/02/04	NEW	
1234500013	Z8983900013	2005/01/27	2004/12/01	OVERDUE	
1234500014	Z8983900014	2005/04/10	2005/02/04	NEW	
1234500015	Z8983900015	2005/04/11	2005/01/03		

1/1

Figure 1: RFQ iView Example – RFQ Quote Maintenance



Part Detail | Component Costs | Tooling | Material Information | Quality Information | Transport & Packaging Information | Engineering Information

Jigs, Tools & Checking Equipment

	Item	Description	Quantity	Date of Delivery	Amount
☐	00020	Tool 1	1.000	2006/07/31	643,410.00
☐	00030	Tool 2	1.000	2006/02/10	421,450.00
☐	00040	Tool 3	1.000	2006/02/30	392,300.00
☐	00050	Tool 4	1.000	2006/03/30	92,500.00

1/1

Delete | Add New

Total Tooling (excl.VAT): 1,549,660.00 | Recalculate Total

Misc Information

Tooling manufacture & tryout lead time (weeks): 100

Guaranteed tool life (pieces): 350,000.00

Save To SAP | Cancel

Figure 2: RFQ iView Example – RFQ Tooling Information

The “RFQ iView” was complex enough to make a good comparison between Java and BSP. Code snippets and observations follow.

Finding the Right Technical Skills

When trying to source SAP Portal developer skills, one can immediately see that there is potentially a much larger pool of Java developers out there than ABAP developers. This is mainly due to the large output from universities that have a passion for Java nowadays. However, this assertion is relative, and hence debatable, because the vast majority of Java developers have no (or only little) experience in enterprise application development. They write programs, but not applications, and have difficulties in understanding and communicating practical business issues.

In addition, the practical applications written in Java are limited and concentrate mostly around portal developments or smaller application add-ons. This is also mirrored on the job markets. While there is still a high demand on enterprise level ABAP developers, the opportunities for Java developers are still a niche market.

Language Complexity

Having made the transition from Java to ABAP, we can honestly say that Java is a far more complex language than ABAP. ABAP's syntax (which stems from the more readable 3rd generation languages like PASCAL, COBOL, and dBASE) is relatively simple to pick up, and you get a much better understanding of the underlying business processes at the same time.

Time is money in any business environment. When developing a Java iView for the SAP Portal, you would typically need TWO developers for any one application—one Java developer for the front-end, and one ABAP developer for the back-end.

From a business point of view, it is not a very comfortable situation needing TWO developers to solve ONE problem. It is not economical, as you block two widely redundant resources, and it is often awkward as the two developers need to communicate a lot with each other to synchronize their work. (To many, however, this would seem fine, as either programmer's work should be totally transparent to the other.) In reality, this synchronization never happens to a satisfying degree, leading in tradeoffs regarding the quality of work. When using BSP development, you have a tool on hand that allows you to achieve the same goal with only one developer, thus circumventing the above mentioned difficulties.

Reality may teach another lesson: the only efficient and successful path for integrated development is having skilled developers who understand both worlds – ABAP and JAVA – equally well.

Think about the "potential" application time needed for Java iView development: meetings with both developers; specifications for both developers; development and testing for both developers; and obviously keeping both developers skilled and trained in their respective technologies. These costs in time, effort, and maintenance can quickly add up.

Using the Java Connector (Jco) library[3] to communicate from Java with a disparate SAP system is very well documented and easy to use. The problem is that there are two developers needed to get a *relatively simple* Java iView onto the SAP Portal. Different development speeds, and higher expectations from either side, can slow down application development pretty dramatically, and require a high amount of patience to avoid animosities and rivalry. From a Java developer's perspective, receiving export parameters like LIFNR, EBELN, etc., from a function module in R/3 can be pretty confusing as first. Furthermore, figuring out how the application fits into the underlying business process can be pretty difficult to grasp.

ABAP is Easier to Learn

Conversely, it is a pure frustration for an ABAP, PASCAL, ADA, or Visual Basic developers when they encounter the inflation of parenthesis and curly brackets, which still dominate the overall layout of a Java class.

[3] The Java Connector JCo had been originally developed by Thomas Schuessler of arasoft.de in the later 1990s

With ABAP and BSP development, you have ONE developer doing the entire application. The understanding of the business process from a developer's perspective cannot be stressed enough. This is definitely one of the big advantages found in BSP development, in that ABAP developers can utilize business process knowledge already gained from years of "traditional" experience.

So the conclusion is, as long as enterprises are driven by R/3 (and hence ABAP – back-end servers), a considerably elevated degree of ABAP skill is indispensable. This implies that you can do your future work in Java with a great deal of ABAP know-how or you find a way to continue fulfilling your work in ABAP, a way offered through BSP developments.

When looking at what a programmer needs to do in order to get a basic SAP Portal application up and going, it soon becomes obvious that ABAP is the more attractive choice. Let's take a closer look at this.

Syntax

ABAP syntax is much easier to read and learn than Java! Most of the SAP developers out there with a little Java knowledge will agree that the lines of code to open a file and output it to the screen, for example, are easier to read and learn in ABAP than in Java.

Let's take a look at some code:

```
data: lv_filename(128) TYPE c default '/usr/tmp/myFile.txt',

      lv_buffer(128)   TYPE c.

…

open dataset lv_filename for input in text mode.

….

do.

  read dataset lv_filename into lv_buffer.

  if sy-subrc <> 0.

    exit.

  endif.

  write / lv_buffer.

enddo.

…
```

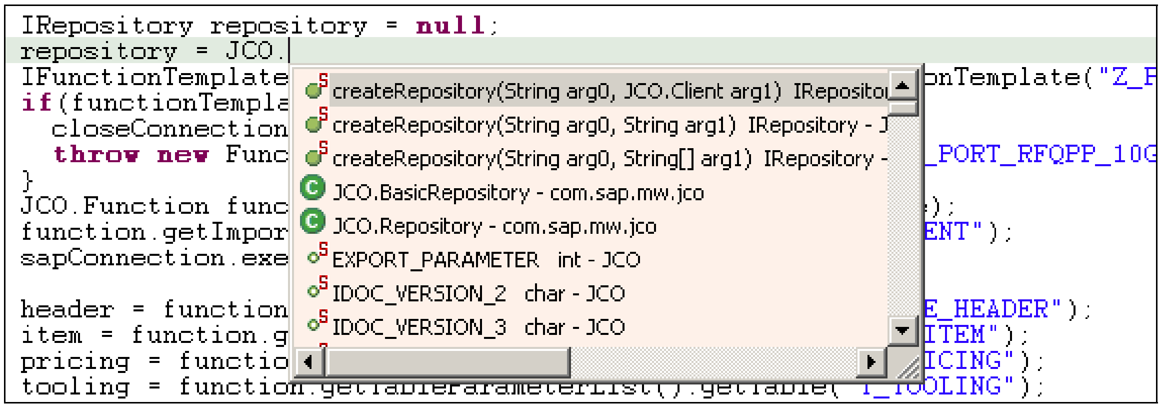
Figure 3: Example of ABAP Code

```
…
private static final int BSIZE = 1024;
…
FileChannel fc = new FileInputStream("myFile.txt").getChannel();
ByteBuffer buffer = ByteBuffer.allocate(BSIZE);
fc.read(buffer);
buffer.flip();
while(buffer.hasRemaining())
  System.out.print((char)buffer.get());
```

Figure 4: Example of Java Code (Typical Scenario Using java.nio.* Package)

From our experience in BSP development, we have found that we need to do substantially less code in ABAP than Java to get the same functionality working on the SAP Portal.

One big advantage of Java development using Eclipse is the "intellisense" feature provided when writing code that displays the possible properties and parameters while you type. ABAP is not at that stage yet, and you often have to go to transaction SE24 just to see what properties, methods, (and associated parameters) can be used with a certain object. You can use "pattern" but it is not the same as a nice "drop down" list box of available properties and methods.



```
IRepository repository = null;
repository = JCO.|
IFunctionTemplate                                 onTemplate("Z_F
if(functionTempla
  closeConnection                                 _PORT_RFQPP_10G
  throw new Func
}
JCO.Function func                                 );
function.getImpor                                 ENT");
sapConnection.exe

header = function                                 E_HEADER");
item = function.g                                 ITEM");
pricing = functio                                 ICING");
tooling = function.getTableParameterList().getTable("T_TOOLING");
```

```
createRepository(String arg0, JCO.Client arg1) IReposito
createRepository(String arg0, String arg1) IRepository - J
createRepository(String arg0, String[] arg1) IRepository -
JCO.BasicRepository - com.sap.mw.jco
JCO.Repository - com.sap.mw.jco
EXPORT_PARAMETER  int - JCO
IDOC_VERSION_2  char - JCO
IDOC_VERSION_3  char - JCO
```

Figure 5: Intellisense Feature (to Help with Typing Code in Eclipse)

Database Access

One of the main strengths of ABAP is its database access layer that is an integral part of the programming language itself, and models a dialect of the open SQL standard. This, combined with the tightly integrated development environment (repository, data dictionary, etc.), frees the developer from all database connection issues typical in most application development environments.

In ABAP the developer is not required to care for any "open connection" / "close connection" statements, as shown in Figure 6.

```
CALL FUNCTION 'Z_RFQPP_GETDETAIL' DESTINATION lv_dest
      EXPORTING
        document             = lv_ebeln
      IMPORTING
        …
      TABLES
        …
```

Figure 6: Illustration: Database Access in ABAP

Java on the other hand needs to explicitly handle the connections to SAP R/3. Figure 7 demonstrates a typical code example of opening a connection to SAP using connection pooling.

```
…
IJCOClientService clientService =
   (IJCOClientService)request.getService(IJCOClientService.KEY);
poolEntry = clientService.getJCOClientPoolEntry("SAPProd500", request);
sapConnection = poolEntry.getJCOClient();
…
sapConnection.connect();
…
try{
  sapConnection.execute(function);      //e.g. 'Z_RFQPP_GETDETAIL'
}catch(JCO.AbapException ex) {
…
```

Figure 7: Illustration: Database Access in Java

One could argue that the Java “connection issues” (compared to ABAP) will definitely improve in the future with SQLJ (OPEN SQL for Java). This is currently a major feature of SAP Web AS 6.40 and looks very promising. How it matches ABAP’s performance and ease of use is still to be seen.

The ideal would be, of course, to make SQLJ part of the Java language specification. It will certainly happen in the context of the upcoming standard for Java Data Objects (JDO). JDO (similar to .NET’s Active Data Objects) introduces a data access layer that presents any data provider as a fully self-contained object and cleans the path for pure object databases.

Connections.

As object databases are still in their adolescence, we need to concentrate on what we find here today. Java uses the concept of connection channels to connect to a database. Using connections over an integrated database provides flexibility but makes prototyping more difficult.

While seeing the above Java code for the first time could make you shiver, you should, however, remain fair and realistic. The need to maintain database connection channels in Java provides for greater flexibility, as it allows connecting to any kind of data source. On the other hand, rapid prototyping and agile development become slightly more difficult, due to the missing integration of the database layer into the IDE. However, in real development situations, this is not too high of a challenge. In an application development, you would probably create and open the connection object upon creation of the application object in its constructor code. This will leave you with a "ready to use" database connection that you can use pretty similarly to the intrinsic ABAP.

Using database connections compared with the fully intrinsic database has, of course, advantages as well. One advantage is that a database connection does not necessarily have to be a real physical database, but could well be a virtual interface into some remote Web service. Imagine AMAZON.COM as a huge bibliographic catalog that you could simply access via SQL, provided someone wrote a driver that translates the SQL statements into the AMAZON Web service language.

The true value of the intrinsic OpenSQL language in ABAP is the integration into the programming language syntax, and that it is interwoven with the data dictionary. Therefore the compiler can check whether table or column names are actually correctly spelled and exist at all. In the usual Java scheme, such errors are only detected at execution time, thus eventually leading into a run-time-error and a program crash upon execution of the statement.

Libraries

Java provides more libraries and associated functionality than ABAP does. While this is a definite plus in favor of Java, it is immediately diminished by the language constraints that make using the libraries rather awkward. The Java developer does have to take extra care to download the appropriate JAR files (if needed), ensure that the classpath environment variable is set correctly, and use the correct import statements in code in order to use certain classes/interfaces, etc. IBM's Eclipse does, however, take much of this effort off of the developer's shoulders. When it comes to SAP Portal development, you would like to keep overall complexity and maintenance to an absolute minimum.

Although a Java developer is provided with a great Portal development environment using IBM's Eclipse framework (eclipse.org) and associated Portal Development Kit (PDK), there are still "unnecessary extras" that need to be looked at and maintained, compared to BSP development. In any one Java source file, you could find numerous library import statements (as a tribute to flexibility) that do add increased complexity to the application as a whole. This can be seen in Figure 8.

```
….
import com.sapportals.portal.prt.component.IPortalComponentProfile;
import com.sapportals.htmlb.rendering.IPageContext;
import com.sapportals.htmlb.type.AbstractDataType;
import com.sapportals.htmlb.enum.DataType;
import com.sapportals.htmlb.table.TableView;
import com.sapportals.htmlb.table.DefaultTableViewModel;
….
….
AbstractDataType rowEdited = AbstractDataType.getInstance(DataType.STRING);
….
```

Figure 8: Illustration: Library Usage Needs to Be Declared in JAVA

ABAP/BSP on the other hand does not require explicit library import statements to be able to declare and/or use variables, etc., of a specific type, interface, or class. How it works can be seen in Figure 9.

```
DATA: lx_my_view TYPE REF TO if_bsp_page.
…
FIELD-SYMBOLS: <fs> TYPE ihttpnvp.
…
```

Figure 9: Illustration: DDIC Libaries Are Always Known in ABAP Without Declaration

The role of the import statements in ABAP is fully delegated to the data dictionary, which records any object that exists in the same ABAP instance. Such a repository would certainly be a priority wish for future enhancements to Java. NET defines the libraries in external manifests so that the compiler defines contexts. This approach is even more flexible than ABAP, while it retains the full adaptability of the library use of Java.

Single-Sign-On

One of the focus points of the SAP Portal is to allow seamless interaction with multiple applications through one, central "unified" framework.

For a BSP developer, the interaction to R/3 is seamless. Should the situation occur where BSP development occurs on another server making RFC calls into the R/3 back-end; it is up to the "SAP Portal administrator" to set up the security certificates correctly to enable SSO.

For a Java developer, the SSO side of things is very well documented and relatively simple to implement. There is, however, more effort required by the developer to implement this correctly. Figure 10 shows an example of "standard" code required to enable SSO into the R/3 back-end.

```
import com.sapportals.portal.prt.component.IPortalComponentRequest;

….

private IPortalComponentRequest request;

….

request = (IPortalComponentRequest) getRequest();

….

poolEntry = clientService.getJCOClientPoolEntry("SAPProd400", request);

….
```

Figure 10: Illustration: Single SignOn in Java

Session Handling

Most of the SAP Portal applications we have developed did require that the objects have "*session state*". This is handled quite differently in Java compared to BSPs. The "Model-View-Controller" (MVC) design pattern plays a big role in providing a nice framework for Portal development and directly influences how "*state*" is handled, in either Java or BSPs.

Session Handling in JSP

The Java implementation for handling "*session state*" is relatively simple to implement, but does require a lot of object instantiation and manipulation. Figure 11 shows the basic interaction among the Java "JSPDynPage", the JSP page, and the object itself.

```
….
private IPortalComponentRequest componentRequest;
private IPortalComponentSession componentSession;
….
public void doProcessAfterInput() throws PageException {
…
  componentRequest = (IPortalComponentRequest) getRequest();
  componentSession = componentRequest.getComponentSession();
…
  myPortalObject = (PortalObject) componentSession.getValue("myPortalObject");
….
  //Manipulate object's properties etc are necessary…
….
  componentSession.putValue("myPortalObject", myPortalObject);
…
}
…
public void doProcessBeforeOutput() throws PageException {
….
  setJspName(jspPage);
….
}
```

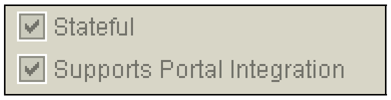
Figure 11: Illustration: Code for Persistent Session State in Java: JSPDynPage

```
…
<jsp:useBean id="myPortalObject" scope="session" class="com.portal.PortalObject" />
…
<hbj:textView id="txtVendor" text="<%=myPortalObject.getVendorName()%>"
   design="LABEL"/>
…
```

Figure 12: Illustration: The JSP Page for the JSPDynPage Example

Persistence Session Handling in BSP

To get the same effect in BSP, we do it a little differently. Out of pure habit, we have always ensured that both the “Stateful” and “Supports Portal Integration” check boxes are ticked in the BSP applications properties. This is depicted in Figure 13.



☑ Stateful
☑ Supports Portal Integration

Figure 13: BSP Application Properties

You do need to beware of abusing this feature too often, as it appears obvious that the session state needs to be stored somewhere on the BSP server, and so it does. If you have many simultaneous requests, it may overflow your session state cache.

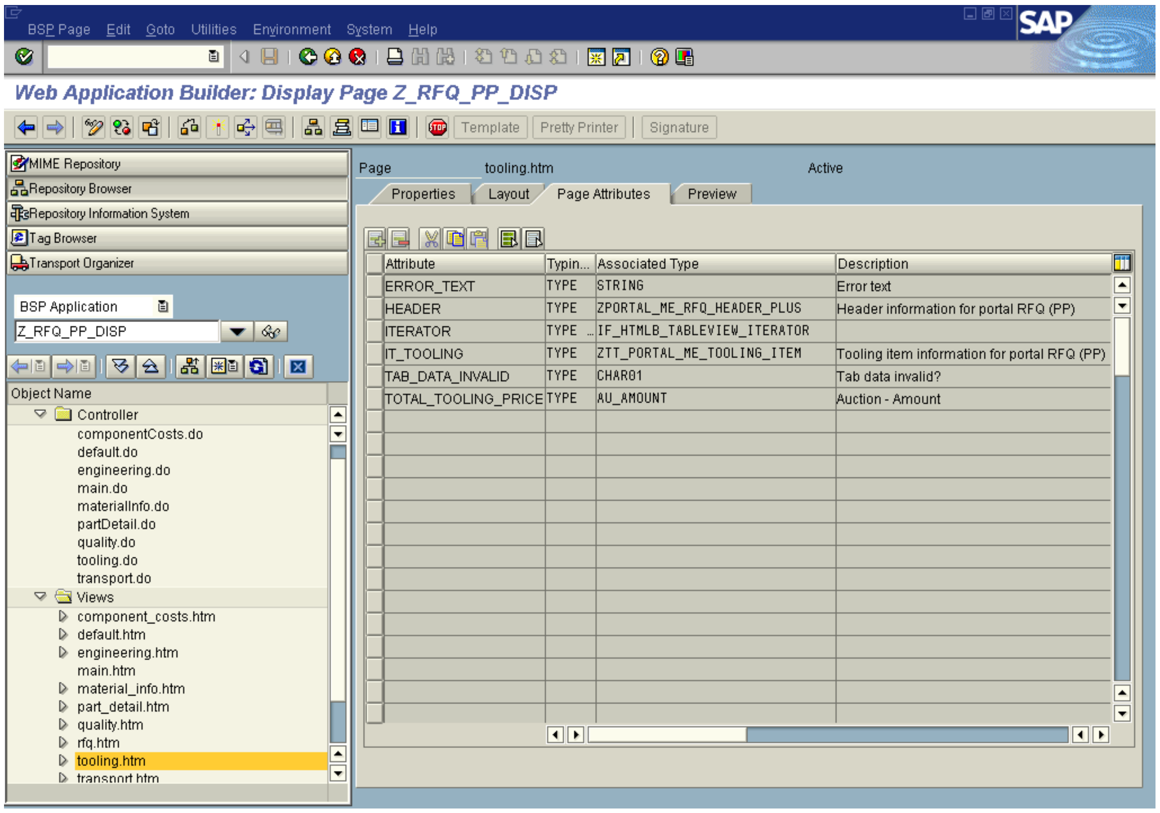
In Java nearly everything is an object[4] and hence the JSP page needs to (typically) reference an object in “session state”, in order to read its properties, etc., and display them on the Web page. In ABAP, not everything is an object and, therefore, the developer can just pass through the variables that are needed on a specific “view,” and not the whole reference variable. This is shown in the following:

```
METHOD do_request.

  DATA: lx_tooling_view TYPE REF TO if_bsp_page.

….

  lx_tooling_view = create_view( view_name = 'tooling.htm' ).

  lx_tooling_view->set_attribute( name = 'TOTAL_TOOLING_PRICE'

                                  value = lv_total_tooling_price ).

….

  call_view( lx_tooling_view ).

ENDMETHOD.
```

Figure 14: Controller Class

A BSP developer is given a very nice maintenance screen when creating a “view” in a BSP application. There he or she can set the relevant import parameters and associated data types, etc., to be used by the view. This is depicted in Figures 15 and 16.

[4] Java and ABAP (also: DELPHI, .NET, ADA) are – other than SMALLTALK or LISP - object aware but not bottom-up object oriented. A fully object-oriented language should be extensible by itself. Neither Java nor ABAP allows this, e.g., you cannot redefine or overload intrinsic operators like assignment (=) or arithmetics (+,-,*,/).



Web Application Builder: Display Page Z_RFQ_PP_DISP

BSP Application: Z_RFQ_PP_DISP

Page: tooling.htm Active

Attribute	Typin...	Associated Type	Description
ERROR_TEXT	TYPE	STRING	Error text
HEADER	TYPE	ZPORTAL_ME_RFQ_HEADER_PLUS	Header information for portal RFQ (PP)
ITERATOR	TYPE ...	IF_HTMLB_TABLEVIEW_ITERATOR	
IT_TOOLING	TYPE	ZTT_PORTAL_ME_TOOLING_ITEM	Tooling item information for portal RFQ (PP)
TAB_DATA_INVALID	TYPE	CHAR01	Tab data invalid?
TOTAL_TOOLING_PRICE	TYPE	AU_AMOUNT	Auction - Amount

Object Name
- Controller
  - componentCosts.do
  - default.do
  - engineering.do
  - main.do
  - materialInfo.do
  - partDetail.do
  - quality.do
  - tooling.do
  - transport.do
- Views
  - component_costs.htm
  - default.htm
  - engineering.htm
  - main.htm
  - material_info.htm
  - part_detail.htm
  - quality.htm
  - rfq.htm
  - tooling.htm
  - transport.htm

Figure 15: Illustration: Maintaining Session Variables in BSP

```
….
<htmlb:inputField id      = "TOTAL_TOOLING_PRICE"
                  type    = "BCD"
                  width   = "120"
                  disabled = "true"
                  value   = "<%= TOTAL_TOOLING_PRICE %>" />
```

Figure 16: Illustration: BSP Page Using Persistence Variables

Although both options solve session state “problems” in their accompanying MVC architecture quite well, the BSP approach appears to be more sexy than the Java implementation, purely due to flexibility and general ease of use.

Validation

Experience shows that the amount of time a programmer spends on trying to recuperate from incorrect entry of data consumes a **large** amount of the overall development time.

The aim of a “*development framework*” should be to minimize the amount of time spent by the developers on validation code, so that they can spend more time on the more important “*business issues*”.

You soon realize that it would take far less time to build great validation processes throughout the entire BSP application, than it would to do the same in Java. For example, let's say you have an HTMLB TableView control on the "view" that allows the user to input data into a "quantity" field (decimal value). Let's also say that this field is to be rendered as "*user*".

Easy Data Entry Validation in HTMLB

In BSPs all you need to do is demonstrated in the code in Figures 17 and 18, in order to have proper (built-in) validation on the field. All accompanying JavaScript is generated for you in the background.

```
<htmlb:tableView id = "Tooling"
             design          = "ALTERNATING"
             ….
             iterator        = "<%= ITERATOR %>"
             table           = "<%= IT_TOOLING %>"
             width           = "100%" >
….
<htmlb:tableViewColumn>
<htmlb:tableViewColumn columnName = "QUANTITY"
                        width     = "80"
                        type      = "user"
                        title     = "Quantity"
              horizontalAlignment = "RIGHT" >
</htmlb:tableViewColumn>
….
</htmlb:tableViewColumns>
</htmlb:tableView>
```

Figure 17: Illustration: Data Entry Validation in ABAP BSP

```
METHOD if_htmlb_tableview_iterator~render_cell_start.

….

  CASE p_column_index.

….

     WHEN lc_quantity_col.  //Quantity column

         ….

         lv_quantity = <fs_tooling>-quantity.

         p_replacement_bee = cl_htmlb_inputfield=>factory(

              id = p_cell_id

              value = lv_quantity

              type  = 'BCD'

              dovalidate = 'TRUE'

              width = '120'

              alignment = 'right'

              decimals = '2').

  ENDCASE.

ENDMETHOD.
```

Figure 18: Illustration: Controller Class to Execute the Data Entry Validation in ABAP

You can see that it does not take much to get awesome validation on a field in BSPs. You can get a similar effect in Java, but it requires more effort. It is actually very surprising how easy it is to implement validation in BSPs—it certainly saves a lot of time!

The HTMLB extension library houses all of the user controls generally displayed visually to the user on the SAP Portal. There are still some compatibility issues between the Java and BSP versions of the HTMLB framework. There are a couple of features found in HTMLB (BSP) that we did not find in the Java library, but this can be expected to change as the libraries evolve.

One thing you may quickly notice is that the Java HTMLB InputField control does not have the “doValidate” property.

If you go to the “Class Builder” (transaction SE24) in SAP and type “CL_HTMLB_INPUTFIELD”, and view its properties and methods, and then view the JavaDocs for the “com.sapportals.htmlb.InputField” class, you will see that they are quite different in implementation.

Javascript is No Replacement for HTMLB

You may argue that certain HTMLB controls have an “onClientClick” event through which you can trigger JavaScript, but once again, not ALL controls have this. We have had to use TabStrip controls a lot, and this is one of the controls that has no “onClientClick” event. This is not a problem for a BSP developer, in that, should the “doValidate” method be set for a specific InputField, the user will not be able to go to another tab until the InputField’s contents are correct. This is, however, a little different in Java. The Java developer has roughly three options:

1. Use a combination of methods like
    - setRequiresValidation(boolean requiresValidation)
    - setValidator(Validator validator) of the “com.sapportals.htmlb.InputField“ class
2. Do server side validation
3. Go through tricky JavaScript generation to get the field validated correctly

The “server side” validation is shown in Figures 19 and 20 and shows the InputField in the TableView with a little red “error” box surrounding it (if incorrect).

```
<%@ page import=" com.portal.ToolingTableCellRenderer, … %>

<hbj:tableView id = "Tooling"

           design = "ALTERNATING"

                        …

            model = "myPortalObject.getToolingModel"
             ----

            width = "100%"

       onNavigate = "onNavigate">

                    <%

       Tooling.setUserTypeCellRenderer(new ToolingTableCellRenderer());

                        ….

                       %>

</hbj:tableView>
```

Figure 19: Illustration: Data Entry Validation in Java JSP

```
public class ToolingTableCellRenderer implements ICellRenderer {

…

public void renderCell(int row, int column, TableView tableView, IPageContext
   rendererContext) {

…

String quantity;

InputField ip;

DecimalFormat df = new DecimalFormat("0.00");

…

  switch(column){

…

    case co_quantity_col:

      ip = new InputField("txtQuantity");

      ….

      quantity = tableView.getValueAt(row, column).toString().trim();

      ip.setString(quantity);

      ….

      try{

        Double.parseDouble(quantity);

        df.parse(quantity);

      }catch(NumberFormatException ex){

        ip.setInvalid(true);

     }catch(ParseException e){

        ip.setInvalid(true);

     }

…

   ip.render(rendererContext);

   break;

  }
…
}
```

Figure 20: Illustration: Class that Executes the Data Entry Validation in Java

Bear in mind that the developer still needs to write code too. For instance, you will want to keep the user on the current tab and display a suitable error message. All these "extras" add unwanted time to the development cycle. And if the developer wishes to try the "JavaScript generation" route, then this would look similar to the code in Figure 21, for a basic field on the screen, not to mention what a "TableView JavaScript routine" would look like!

```
….

<script language="javascript">
  function validateVendor(){
   if(document.mainform.txtVendorNumber.value == ""){

   alert("Please enter a vendor number.";
     return false;
    }else{
     return true;
    }
  }
  </script>
  ….

<hbj:button id = "myButtonID"
      onClick = "onButtonClicked"

onClientClick = "if(!validateVendor())htmlbevent.cancelSubmit = true;"
                    text = "Go" />

….
```

Figure 21: Illustration: Data Entry Validation with Hand-coded JavaScript

So, in the end, validation can be handled equally well in Java and BSP, but we feel the BSP version requires less object manipulation and coding effort and allows for much easier support and error tracking when unwanted effects appear during runtime in a production environment.

Development Environment

As mentioned previously, the integrated ABAP Workbench, repository, and data dictionary are really hard to beat. IBM Eclipse is a pleasure to work in, but still has some way to go in order to match SAP in this arena, and .NET is a respectable challenger who is following close behind[5]

The main weaknesses of Eclipse compared to the ABAP Workbench are the power of the debugger and, especially, the version control and change management. It is interesting to see that those "going productive" support tools are mistakenly ignored by the challengers of ABAP.

Another feature not found in any of the major development studios is the ABAP Object Navigators that hyperlink all tokens from the source code in the editor's window, and allow you to "jump" from

5 We found a good decision matrix to compare the benefits of the individual approaches of .NET, Java, and BSP to design Portal applications in an article fragment on SDN: sdn.sap.com from Ajay Bhardway: Selection of EP Content Development Environment

a variable to its declaration, and from there to the repository, etc. This is done simply by double-clicking on the token's name. This feature is so utterly helpful, not just for those who program in ABAP every day, but also for newcomers who are just starting to learn ABAP, as it allows them to easily explore the landscape and the associations between objects.

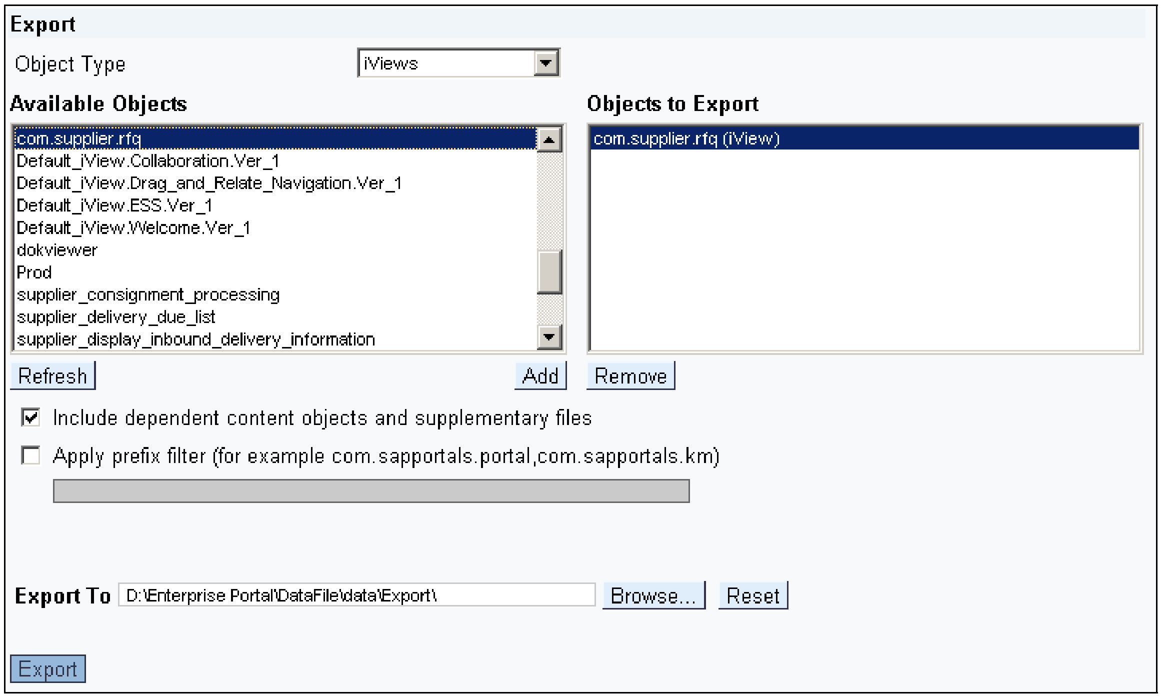
Transport and Version Management

SAP is well known for its reliable change management system known as the "*Transport (and Versioning) Management System (TMS)"* This is something that many ABAP developers take for granted in a big way (compared to what is typically required in other application frameworks).

The transport of a BSP application is exactly the same as any normal SAP transport and is "transparent" to the developer. Version management is automatically handled by SAP and once again is "hidden" from the developer.

The situation is a little different for Java development. The full transport and version management of the Java code needs to be handled explicitly by the Java developer and/or SAP Portal administrator. This definitely adds complexity and overhead in the development process.

It is easy to upload a PAR file from DEV->QA->PROD, but this is generally not the way it should be done. One way of doing this correctly would be to transport the page(s), iView(s), etc. from the DEV (or QA SAP Portal) through to the PROD Portal. This illuminates the need, for example, for .class files to be compiled on the PROD server, etc. This is typically done by the SAP Portal administrator and is depicted in Figure 22.


Export

Object Type: iViews

Available Objects:
- com.supplier.rfq
- Default_iView.Collaboration.Ver_1
- Default_iView.Drag_and_Relate_Navigation.Ver_1
- Default_iView.ESS.Ver_1
- Default_iView.Welcome.Ver_1
- dokviewer
- Prod
- supplier_consignment_processing
- supplier_delivery_due_list
- supplier_display_inbound_delivery_information

Objects to Export:
- com.supplier.rfq (iView)

[✓] Include dependent content objects and supplementary files

[ ] Apply prefix filter (for example com.sapportals.portal,com.sapportals.km)

Export To: D:\Enterprise Portal\DataFile\data\Export\

Figure 22: SAP Portal iView Export Procedure

You will need to (potentially) use this deployment procedure regardless of whether your iView was written in Java or BSP.

Post-Deployment Adaptation

The real challenge for all development projects comes when the software is ready to be deployed to a production environment. Let us review an example of this. There is often the requirement, for example, that if a file is uploaded into SAP from the SAP Portal, that file should be archived somewhere for auditing reasons. Should you also want to display a URL (to a file created on the Portal server) you may also need to change port numbers and protocols (HTTPS), etc., depending on which server the PAR file is running.

The SAP Portal developer and SAP Portal administrator obviously want the entire transport system on the Portal to be as seamless as possible. They should not have to ever modify "config files" manually after transporting! In order to make the PAR file somewhat "intelligent", you may have to incorporate some XML into the picture. Going back to the "upload file issue", take a look at the example XML file attached to the PAR file on upload in Figure 23.

```
<?xml version="1.0"?>
<config>
  <portalserver>
    <servername>SAPPROD400</servername>
    <protocol>https</protocol>
    <portnumber>80</portnumber>
    <filepath>c:\\SAPPortal\\Docs\\</filepath>
    <virtualdirectory>/SAPPortal/Docs/</virtualdirectory>
   <archivepath>c:\\SAPPortal\\Docs\\Project_1_Upload_Archive\\</archivepath>
  </portalserver>
….
….
</config>
```

Figure 23: Illustration: Server_Properties.xml

The above file could obviously have any number of <portalserver> entries, and depending on which server the PAR file is executing on, it would react appropriately. This works very well in the production environment, but does increase the amount of work on the Java developer side. Figures 24 through 26 show the work that would normally go into getting this seamless XML transport working.

```
….
XML_File_Name.value=server_properties.xml
….
```

Figure 24: Illustration: *default.properties*

```
public void doInitialization(){

….

  xmlFileName = componentProfile.getProperty("XML_File_Name");

….

  try{

   aResource = componentRequest.getResource(IResource.XML,"xml/" +
   xmlFileName);


   myPortalXMLObject.readXMLFile(aResource.getResourceInformation().getSource())
   ;  }catch(ResourceException e){

  …

  }

}
```

Figure 25: Illustration: *JSPDynPage*

```
public synchronized void readXMLFile(String filePath, String serverName) throws


   ServerNotFoundException,ParseException,XMLException{

….

DocumentBuilderFactory docBuilderFactory = new DocumentBuilderFactoryImpl();

DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();

Document doc = docBuilder.parse (new File(filePath));

…

NodeList listOfPortalServers = doc.getElementsByTagName("Portalserver");

totalPortalServers = listOfPortalServers.getLength();

….

while((i < totalPortalServers) && (serverFound == false)){

  ….

  tmpServerName = ((Node)textServerList.item(0)).getNodeValue().trim();

  if(tmpServerName.compareTo(getServerName()) == 0){

     serverFound = true;

     ….

     tmpArchivePath =
   ((Node)textArchivePathList.item(0)).getNodeValue().trim();

     ….

     setArchivePath(tmpArchivePath);

     ….

}

….
```

Figure 26: Illustration: ***MyPortalXMLObject's readXMLFile Method***

So, although relatively simple code, it is nevertheless extra development overhead that needs to be considered on the Java side.

With regard to "version management", a Java developer has a number of options available. Obviously the "easy" approach is just to archive your PAR Java projects, with the current date and time, in a directory on a server. This is definitely **not** the best approach. Another option is to use a product like WinCVS to aid in proper version control and management. The developer obviously has to then worry about checking in and checking out code all the time. This is definitely acceptable, but not nearly as efficient as what SAP provides in this regard.

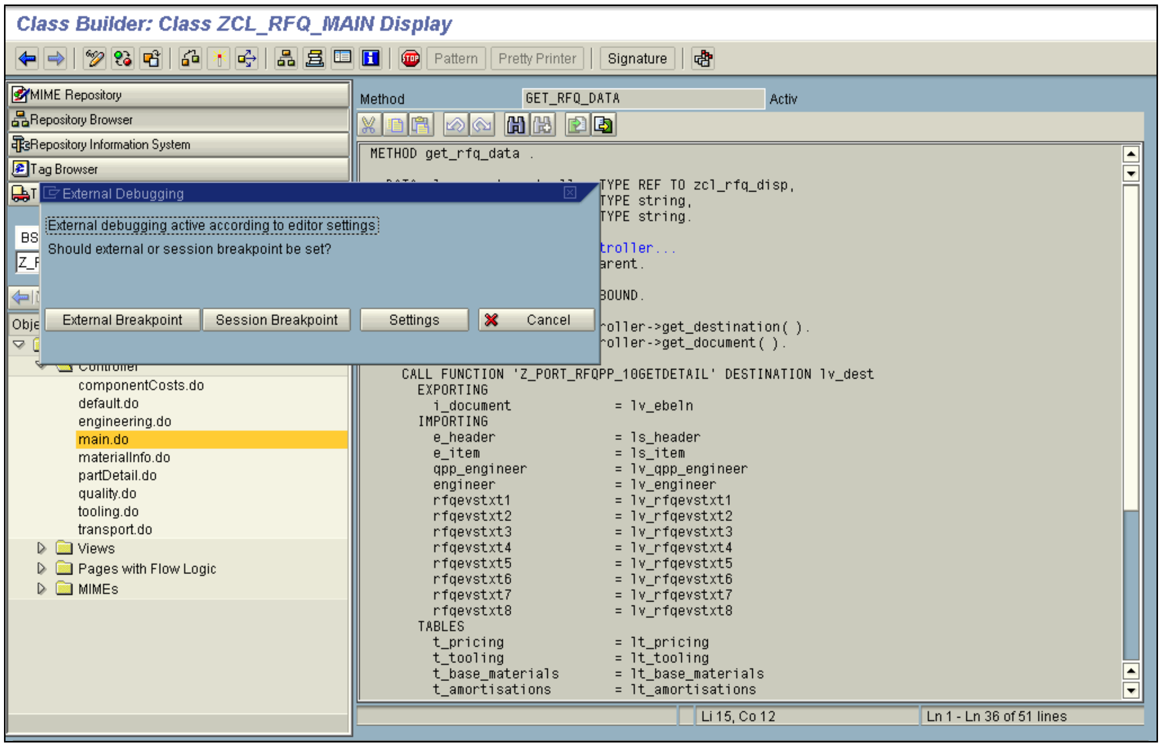
On a positive note, SAP has come up with a “design time repository” for Java development that offers features such as version management and incremental activation procedures that do away with the “nightly build procedures” previously undertaken.

This new development infrastructure is currently very immature but has a positive future ahead.

Debugging

Those coming from the Java world to the SAP environment will honestly say that debugging a BSP application is much easier than in a Java Portal application. Let’s say you arrive at work on a Monday morning and have to sort out a production issue ASAP! Let’s see how the debugging environments compare.

In ABAP, you simply log onto the production Web AS, go to your BSP application via transaction SE80, and set your “External Breakpoint” as shown in Figure 27.



Method GET_RFQ_DATA Activ

External debugging active according to editor settings

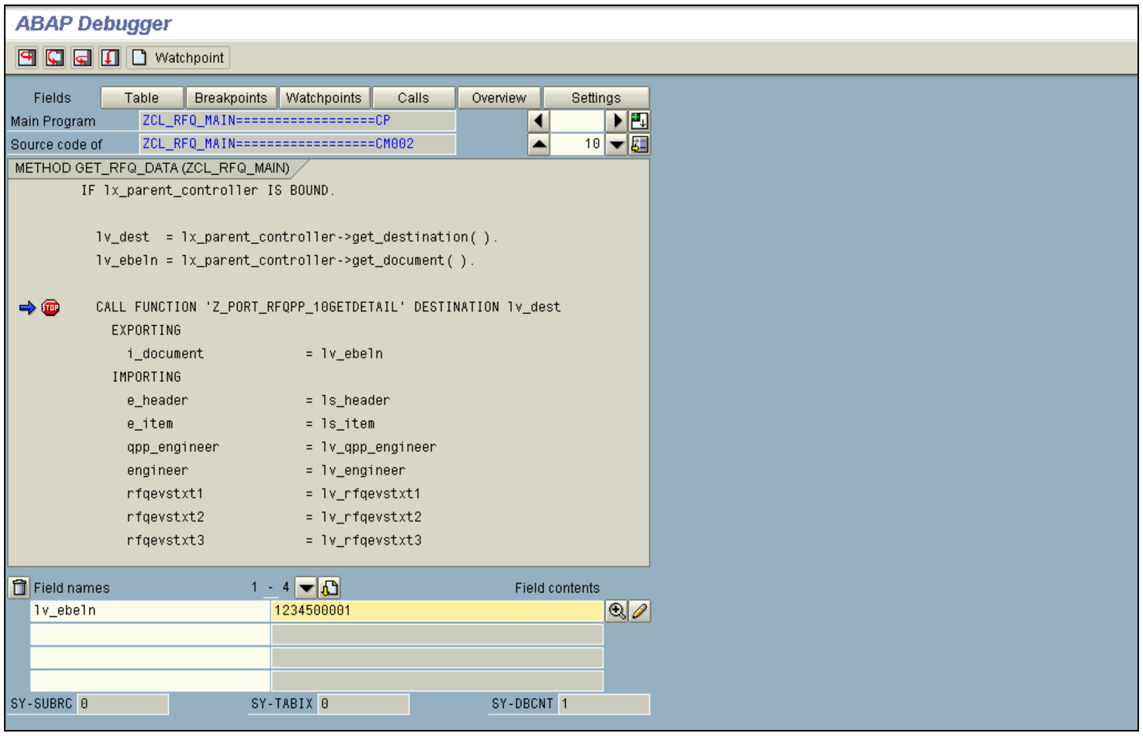
Should external or session breakpoint be set?

```
METHOD get_rfq_data .

                                  TYPE REF TO zcl_rfq_disp,
                                  TYPE string,
                                  TYPE string.

                                  troller...
                                  arent.

                                  BOUND.

                                  roller->get_destination( ).
                                  roller->get_document( ).

    CALL FUNCTION 'Z_PORT_RFQPP_10GETDETAIL' DESTINATION lv_dest
      EXPORTING
        i_document               = lv_ebeln
      IMPORTING
        e_header                 = ls_header
        e_item                   = ls_item
        qpp_engineer             = lv_qpp_engineer
        engineer                 = lv_engineer
        rfqevstxt1               = lv_rfqevstxt1
        rfqevstxt2               = lv_rfqevstxt2
        rfqevstxt3               = lv_rfqevstxt3
        rfqevstxt4               = lv_rfqevstxt4
        rfqevstxt5               = lv_rfqevstxt5
        rfqevstxt6               = lv_rfqevstxt6
        rfqevstxt7               = lv_rfqevstxt7
        rfqevstxt8               = lv_rfqevstxt8
      TABLES
        t_pricing                = lt_pricing
        t_tooling                = lt_tooling
        t_base_materials         = lt_base_materials
        t_amortisations          = lt_amortisations
```

Li 15, Co 12   Ln 1 - Ln 36 of 51 lines

Figure 27: Setting “External Breakpoint” in ABAP Source Code on the SAP Web AS

You then launch the browser and enter “debugging mode”.


ABAP Debugger

Main Program ZCL_RFQ_MAIN=================CP
Source code of ZCL_RFQ_MAIN=================CM002

```
METHOD GET_RFQ_DATA (ZCL_RFQ_MAIN)
      IF lx_parent_controller IS BOUND.

        lv_dest  = lx_parent_controller->get_destination( ).
        lv_ebeln = lx_parent_controller->get_document( ).

        CALL FUNCTION 'Z_PORT_RFQPP_10GETDETAIL' DESTINATION lv_dest
          EXPORTING
            i_document             = lv_ebeln
          IMPORTING
            e_header               = ls_header
            e_item                 = ls_item
            qpp_engineer           = lv_qpp_engineer
            engineer               = lv_engineer
            rfqevstxt1             = lv_rfqevstxt1
            rfqevstxt2             = lv_rfqevstxt2
            rfqevstxt3             = lv_rfqevstxt3
```

Field names	Field contents
lv_ebeln	1234500001

SY-SUBRC 0 SY-TABIX 0 SY-DBCNT 1

Figure 28: Debugging a BSP Application (exactly the same as traditional ABAP)

BSP debugging is as simple as that! Any ABAP code changes will be transported as normal.

In Java, things are a little trickier. You “could” have to go through the following sequence of events:

- Comment out “connection pooling” (if used) and use a “hard-coded” SAP connection in the code.

```
sapConnection = JCO.createClient("300", "username", "password", "EN",
"sap.prod.server", "00");
```

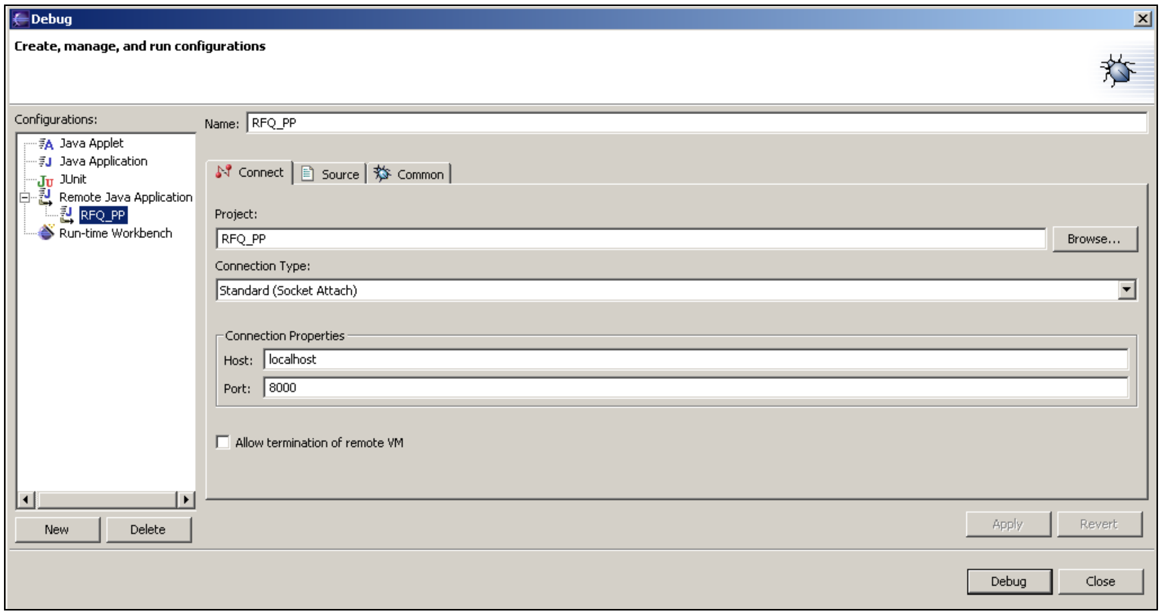
Figure 29: Hard-Coded SAP Connection Example

- Make sure the **TOMCAT** Web server is started on the local machine in “debug mode” using the parameters (or similar depending on version) in Figure 30:

```
rem --- Set TOMCAT_OPTS

set TOMCAT_OPTS=-Djava.library.path=%TOMCAT_HOME%/lib/apps -
   Dhttp.proxyHost=%_PROXY_HOST% -Dhttp.proxyPort=%_PROXY_PORT% -
   Dhttp.nonProxyHosts=localhost -Dcm.decode=X -server -Xdebug -Xnoagent
   -Djava.compiler=NONE -
   Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000

echo TOMCAT_OPTS = %TOMCAT_OPTS%
```

Figure 30: Example Code for Starting TOMCAT Web Server

- Set a break point at the appropriate location in the Java code.
- Run the Portal project in "debug mode" as a "remote Java application" as shown in Figure 31.



Name: RFQ_PP

Project: RFQ_PP

Connection Type: Standard (Socket Attach)

Connection Properties

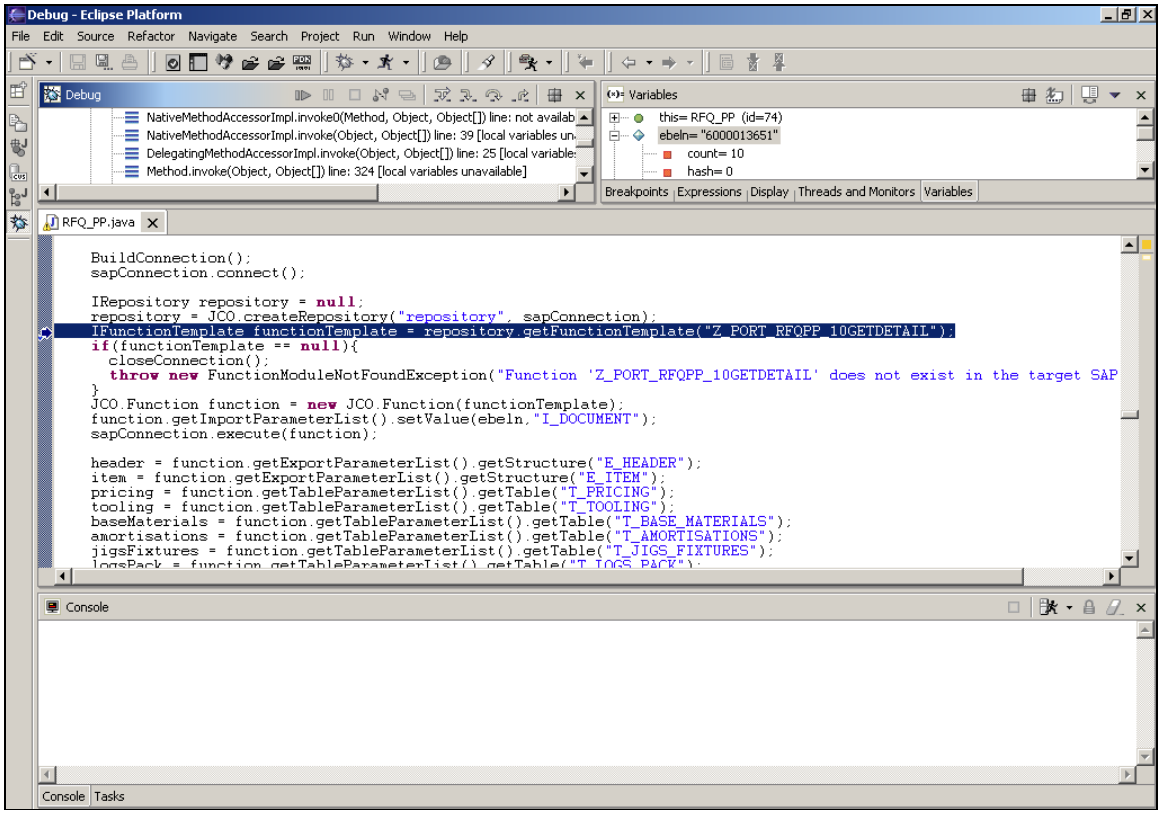
Host: localhost

Port: 8000

☐ Allow termination of remote VM

Figure 31: Running the Portal Project in Debug Mode

- Open the browser to the desired URL and page and initiate debugging by triggering the correct event.
- Debug the application in the Eclipse environment as depicted in Figure 32.


Debug - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Breakpoints Expressions Display Threads and Monitors Variables
RFQ_PP.java
Console
Console Tasks

```
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not availab
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 39 [local variables un
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 25 [local variable:
Method.invoke(Object, Object[]) line: 324 [local variables unavailable]
```

```
this= RFQ_PP  (id=74)
ebeln= "6000013651"
    count= 10
    hash= 0
```

```java
BuildConnection();
sapConnection.connect();

IRepository repository = null;
repository = JCO.createRepository("repository", sapConnection);
IFunctionTemplate functionTemplate = repository.getFunctionTemplate("Z_PORT_RFQPP_10GETDETAIL");
if(functionTemplate == null){
  closeConnection();
  throw new FunctionModuleNotFoundException("Function 'Z_PORT_RFQPP_10GETDETAIL' does not exist in the target SAP
}
JCO.Function function = new JCO.Function(functionTemplate);
function.getImportParameterList().setValue(ebeln,"I_DOCUMENT");
sapConnection.execute(function);

header = function.getExportParameterList().getStructure("E_HEADER");
item = function.getExportParameterList().getStructure("E_ITEM");
pricing = function.getTableParameterList().getTable("T_PRICING");
tooling = function.getTableParameterList().getTable("T_TOOLING");
baseMaterials = function.getTableParameterList().getTable("T_BASE_MATERIALS");
amortisations = function.getTableParameterList().getTable("T_AMORTISATIONS");
jigsFixtures = function.getTableParameterList().getTable("T_JIGS_FIXTURES");
logsPack = function.getTableParameterList().getTable("T_LOGS_PACK");
```

Figure 32: Debugging in Eclipse

- Any code changes require the Portal project to be re-compiled into a new PAR file and uploaded to the Portal server.
- Version control will also need to be handled explicitly.

The SAP Portal administrator would also need to re-transport the Portal page, iView, etc., from a development system to the productive Portal server.

As you can see there is potentially a vast difference in what needs to be done in order to troubleshoot and fix a Java application on the Portal, compared to doing so with a BSP application.

ABAP or Java, Which Is the Best?

We need finally to come back to the initial question: which development environment should you use? It is obvious that none of the frameworks allow easy Portal application development.

What Do We Expect from a Portal Development Environment

Here we should at least lean back for a minute and try to recall what you might expect from a Portal development environment.

- Provide a general application class that plugs the Portal applet into the Portal and takes care of the Single Sign-On
- Provide a global session container that allows you to store session-persistent data
- A high level API for all common GUI elements like menus, entry forms, and tabular lists
- A configurable workflow handler that allows you to define workflow events without programming
- A messaging infrastructure for common tasks like sending SMTP mail, file upload and download, and data entry verification

Generally, we should say that you want to opt for the language where your developers, and more importantly, your support staff, feel most proficient. If you have neither sufficient ABAP nor Java skills, then the steeper learning curve required by Java would speak strongly for ABAP. In cases where you are indifferent towards ABAP or Java, the pendulum will swing unambiguously to favor ABAP, due to its higher stability, easier debugging and maintenance facilities, and better readability.

ABAP and Java Compared

As we reach the end of our paper, let us compare the respective benefits of ABAP and Java with respect to Portal development. The predominant criteria to decide for any development framework can certainly not be driven from the viewpoint of Portal development only. Deployment, maintenance, change management, and security will make big arguments that depend strongly on the individual operating environment and existing IT infrastructure. There are definitely many more areas to compare, but that is not the argument here, so we will look at a short selection of arguments from a Portal developer's view only.

ABAP Merits

- ABAP introduced a fully transparent source-code just-in-time compiler.
- ABAP has an unmatched deployment and version management service, the TMS.
- ABAP allows easy source code debugging (also in production environment).
- ABAP integrates the Data Dictionary into the Development Environment.
- Current BSP versions have more elaborate intrinsic data entry validation methods.
- Currently there are far more ABAP developers with in-depth business process know-how on the market than for any other development environment.

Java Merits

- Java introduced standard algorithms and data-structures through the Design Pattern school.
- Java has an abundance of independent and open source libraries.
- Java provides more features for sophisticated Web effects.
- Modern schools and universities favor Java, and hence the Java community is growing with high speed.

Conclusion

It is clear that SAP has a strategic direction they wish to pursue with Java, and the J2EE stack has got plenty to offer in terms of open-ness and flexibility to the SAP world.

ABAP will continue to rule the back-end, and Java the front-end. Our recommendation would be to use the BSP development in 80% of "general" Portal development and Java in select, justified cases, in that 20% of the development that is hugely complex on the front-end (depending on resources of course). The SAP NetWeaver Developer Studio (Eclipse) is not mature enough to compete with the ABAP Workbench directly, and still has some way to go. SAP is also supporting J2EE with the new Unbreakable Java (also known as "Always On Java") project that promises to put the Java language on top of the unbreakable ABAP engine. With the very "object orientated" approach SAP is taking with ABAP in the future, the "mindset" change from ABAP to Java will hopefully not be too great, and future developers can mix and match the two languages as need be.

After Hours Chat Room

The idea for this article came during some interesting chat room discussions on several Java and Portal issues. The following is a summarized transcript of those discussion showing the most interesting arguments and disputes.

Axel: "Lynton, you are akin to new breed of dedicated Java developers, those who know Java and the development environment by heart. You told me that you however became a great fan of SAP's ABAP development environment. I for my part come from the more traditional tribe of PASCAL and ABAP IV development. Why is it so intriguing with Java from your personal view?"
Lynton: "With Java the possibilities are endless...the ability to code mobile devices, Web Services and even robots that can potentially see, hear and even move is just awesome." "Personally I like the fact that Java is so broad in scope that if you ever got bored of writing Web applications you could quite easily pursue a totally different route in Java like mobile or even game programming. I also like the fact that Java is very focussed around UML, design patterns and layered programming...I feel these concepts are vital to grasp in any developer's mind. All in all getting your hands dirty with programming languages like Java and .NET can be fun...period!"
Axel: "When you speak of endless possibilities: isn't that rather an argument for .NET. I imagine just getting simple RS232 or RS438 devices connected via Java. Finding an appropriate driver others those written for a Solaris machine are simply a nightmare!"
Lynton: "I do agree with you that the amount of money and global backing Microsoft are getting there is just no way that a framework like .NET will fail. From what I've seen it takes a .NET developer far less time to develop an application than it would a Java developer. This, however, may not always be the case. For example, the other day I was playing around on SAP WAS 6.40 and the "drag & drop" features of 'WebDynpro controls' is definitely a step in the right direction! So when I speak of "endless possibilities", yes, I do feel that .NET has a larger array of EASY TO USE device drivers / adapters etc to allow a developer to integrate into many of today's modern applications and devices. But I will remain strong on my personal views that .NET is great for complex front-end applications while Java is at it strongest in the highly transactional backend systems. I feel, for example, that the JCA, JMS standards provided by Java are unmatched in the .NET world. Why do big companies like SAP, Oracle, IBM, Nokia, etc. choose J2EE over .NET? Why are about 95 percent of EAI middleware products J2EE compliant? So in conclusion I think when I talk of these 'endless possibilities' you also need to look at what area of IT you are dealing with...if you want your Nokia cell phone to send a message into SAP I would probably choose the Java route, if you wanted me to make a hugely complex frontend application connected to numerous devices I would probably take the .NET route.....use the language that will handle the current problem best.... even mix and match if you need to. With the way Web Services are going

it's not going to make too much difference it a .NET Web Service talks to a Java Web Service which talks to a ABAP Web Service, all will be transparent to the calling application. Once again I'm not too fussed about syntax or language, I just want to get the job done!!”
Axel: “Why do you like ABAP more than Java then?”
Lynton: “Having done ABAP for some time now, I have realised that the integrated SAP environment is simply hard to beat. Seamlessly integrating into subsequent areas of SAP by simply double clicking on table names, data elements, etc becomes very handy during complex development. The debugging and transport systems of SAP are fantastic...something I feel many ABAP developers take for granted in a BIG way!!”
“Adding onto this, the fact that I'm yet to see an SAP instance crash is also very comforting. I do, however, prefer the new object orientated approach to ABAP than the traditional "top-down" ABAP syntax. With regard to SAP Portal development I can honestly say that it takes me much longer to develop a complex iView in Java than in ABAP / BSP. Java does have many advantages - that's a given, but when it comes to stability and ease of development, ABAP developed on top of WebAS is rock solid.”
Axel: “Do you think Java is easy to learn?”
Lynton: “In a nutshell: no! I feel the main difference between a language like ABAP and Java is that ABAP hides the developer from most of the technical intricacies of the "underlying language". On the other hand you need to be quite technically minded to handle the "bits and bytes" that the Java language can throw at you. I think it also has a lot to do with how passionate the person is and what drive and ambition they have inside them.”
Axel: “For me object programming is a way of thinking that existed ever since, at least the late 1960ties. Therefore I see the current hype on Java relatively critical, as it somehow obstructs the clear view on business related object patterns but rather gives many developers a good excuse to dig into technical and academic issues of object development. What do think? Is Java a step forward or backwards...?”
Lynton: “I'm going to look at this question a little differently. Syntax aside, I do feel that the huge array of current design patterns and "object developments" are actually too overwhelming for the majority of developers out there. When has been the last time you even saw an ABAP program that used a design pattern!! I often here people saying things like "that was probably developed by some super hacker nerd who had too much time on his hands...“ Most developers stick with a few ideas and "business patterns" that they are familiar with in order to get the job done. I do think things have become somewhat "overly complex" in the modern programming world. But I'm not going to sit on the fence here; I will say that Java has provided developers with a "common syntax" and "object understanding" that can easily be moved to another language like C# or VB.NET. I'm not overly concerned about what language you choose between Java and .NET etc...For me it's more the paradigm shift in thinking in patterns and objects.”
“What I would like to see, however, is more focus on making languages like Java much more "Model Driven" (as in the OMG's "Model Driven Architecture"). The dynamic nature of today's world requires a "programming shift" that simply allows developers to manipulate a visual model and see results immediately. SAP's WebDynpro methodology is definitely a step in the right direction in this regard.”
Axel: “Indeed the renaissance of design patterns is something I found also very positive. While caring for the new Java technology many developers came across learning about the positives of reusing software components and sticking to general abstract terms. It also makes discussion about algorithms much easier. But in order to rescue the honour of us old-time programmers I want to mention that design patterns are certainly something that good programmers always abided by - think of such master pieces like "Donald Knuth: The Art of Computer Programming" or the sort version by the inventor of Pascal, Modula and the Mouse, "Niklas Wirth: Algorithm and Data Structures." But this makes me mull over another question: Good programmers have always been a rarity. Will Java produce finally more good programmers?”

Lynton: “I think the number of good programmers will steadily increase over the years. Let's face it, IT is here to stay and people are jumping onto the "IT bandwagon" every day...there are sure to be a couple of good brains amongst them!! What is interesting to see is that people are almost getting sucked into languages like Java and .NET. These languages are basically the "norm" at Universities etc. New systems have also been developed in Java that could have been up and running in a fraction of the time had they been written in languages like Lisp or Python. But we can't choose to implement in these languages because you can't easily find programmers with good enough experience in these languages. This is because they are too busy programming in Java, .NET etc as it's hard to find jobs in Lisp, Python etc. Java is not known as a "hacker language" (and we all know that hacker's have some of the smartest minds in the IT business), BUT it seems that some of these “hackers” are having fun becoming somewhat “familiar” with the Java language just to keep money in their pockets. So yes, I do think Java will produce more good programmers in the future. We can look at this in a slightly different way as well. Designing and making systems seems to be getting easier as the years go by. The OMG's “Model Driven Architecture” is just one of many things that is set to make a programmer's life so much simpler in the future. Who knows, maybe one day we will even be able to talk to our computers and tell them what type of application we would like developed....then couldn't everyone be a good developer?”

Axel Angeli, logosworld.com. *Axel is a senior SAP and EAI advisor and principal of logosworld.com, a German-based enterprise specializing in coaching SAP and EAI project teams and advising IT management on implementation issues. Axel has been in the IT business since 1984, and throughout his career, he has always worked with cutting edge technologies. Axel's SAP experience stems from the good old R/2 days, and he is an expert on SAP’s NetWeaver technology and any kind of ABAP development. A speaker of several languages, Axel specializes in coaching and leading large multi-national teams on complex projects with heterogeneous platforms and international rollouts. Known for his intensive and successful trouble-shooting experience, Axel has been nicknamed by his colleagues as the “Red Adair” of SAP projects. He is the author of the best-selling tutorial “The SAP R/3 Guide to EDI, IDocs, ALE and Interfaces.” Axel’s email address is* axel.angeli@logosworld.com

Lynton Grice *is an application developer contracted to one of the big automotive companies in South Africa. He has a passion for programming and new technologies and takes care of SAP Portal development as well as other SAP EAI requirements. Coming from a typical Java school, he is now just as familiar with ABAP. Lynton’s email address is* lynton.grice@netweaverguru.com