Web Dynpro — what it is, what it does, why it exists, and how to get the best results from it

An introduction to the fundamental principles of Web Dynpro

by Chris Whealy

Chris Whealy
RIG Expert (Web Dynpro),
SAP NetWeaver Regional Implementation Group (RIG) EMEA, Walldorf, Germany

Chris Whealy joined SAP in 1995 as a Basis consultant and ABAP programmer. Shortly thereafter, he turned his attention to Web-based interfaces to SAP systems and began working with the earliest versions of the Internet Transaction Server. Since then, Chris has focused his attention on Web-based front ends for SAP functionality. In 2003, Chris began working with Web Dynpro, both learning the product and writing proof-of-concept applications. Chris is the author of "Inside Web Dynpro for Java" (SAP PRESS, November 2004) and SAP training courses on Web Dynpro.

(Full bio appears on page 60.)

Web Dynpro — SAP's newest user interface (UI) development option for the SAP NetWeaver platform — has been designed to become the de facto option of choice for SAP development. Web Dynpro was created because, like every other software vendor in the Web space, SAP needed a long-term, strategic solution for the many problems faced by Web developers during the implementation of browser-based business applications.

Anyone who has written a Web application of any complexity has probably faced the pain of dealing with applications:

- That work perfectly with one browser, but then develop "unexpected features" when viewed in a different browser
- That require significant rework in the UI layer because the structure of the business data changed
- In which short-cuts were taken during the design phase to get a quick result — only to find that the costs "saved" during development came back to haunt you during the maintenance phase[1]

Web Dynpro is both a development tool for Web-based applications and a runtime environment. It has been designed in such a way that, when used correctly, it can cut the development timescale of a complex business application by as much as 50%. It achieves these dramatic savings by using graphical and declarative programming techniques. The only code developers need to write is that related to the core business process.

The power of Web Dynpro lies in its use of object-oriented (OO) design concepts. Using these concepts, you can create reusable units of business functionality that you can "LEGO-brick" together to form a complete application. The catch? As for all other technologies, you have to understand Web Dynpro and use it wisely to reap its rewards.

[1] This problem is not just confined to Web-based applications.

Note!

Unfortunately, I've seen more than my fair share of very poor Web Dynpro implementations! These were entirely due to a lack of specific Web Dynpro training. Without such training, developers tend to assume that Web Dynpro is "just like any other Web development toolset," which results in applications being built using inappropriate or incompatible design concepts. The resulting applications may well be functional, but the coding is inefficient, performance suffers under high workload, and the maintenance is time consuming, error-prone, and more costly than necessary.

I wrote this article specifically to combat some misconceptions that (for reasons I cannot explain) seem to be quite prevalent among developers concerning Web Dynpro — in particular, what it is, what it does, why it exists, and how to get the best results from its features. I suspect many times that it's the age-old thought process of:

- *"Things I don't understand are complicated."* followed by…
- *"I don't understand Web Dynpro."* therefore…
- *"Web Dynpro is complicated."*[2]

I'm very pleased to tell you that this conclusion is completely false. Web Dynpro is built on simple principles. Understand the principles, and you understand Web Dynpro. (See the sidebar on the next page for some background history.)

In this article I describe the fundamental principles of Web Dynpro using clear, simple language, and illustrate them using an actual implementation[3] built using Web Dynpro for Java. Irrespective of whether you are interested in the Java or ABAP flavors of Web Dynpro,[4] the design principles and final application architecture we'll discuss here are independent of the implementation language. This knowledge will enable you to make a well-informed decision about the usefulness of Web Dynpro at your company and to determine how you can use it to create low-maintenance, high-performance business applications. You'll also come away with the knowledge needed to assess the competence of a potential implementation partner if you outsource your development.

Note!

Everyone can benefit from the information in this article. Even the most seasoned developers can learn the "truth" about this product. The fundamental building blocks of Web Dynpro applications that I cover later in the article will help you understand the design of an actual Web Dynpro application (i.e., the SAP IDES flight model). Although this last section is more suitable for developers (because it goes into the area of software architecture), everyone should read it because it will help you see the possibilities and potential that lie ahead with an implementation based on Web Dynpro.

So what is Web Dynpro really?

Many myths surround Web Dynpro. Before explaining the history and design concepts of Web Dynpro, I want to dispel some of the most common myths.

[2] This type of reasoning is known as an "invalid syllogism." Another example would be to say, "A cat is a mammal, and a dog is a mammal; therefore, a cat is a dog." Hmmmm, really?

[3] The example used in this article is the flight model from the SAP IDES training system.

[4] The two flavors of Web Dynpro are Web Dynpro for Java and Web Dynpro for ABAP. Java Web Dynpro applications are written with SAP NetWeaver Developer Studio, which is a stand-alone development environment based on Eclipse. ABAP Web Dynpro applications, however, are written using the ABAP Development Workbench (transaction SE80) on SAP systems with a 7.00 kernel or higher. End users will have no idea which language was used to write the application since both look identical when displayed on the same client.

The origins of Web Dynpro

The seeds of Web Dynpro were bouncing around within SAP as early as the beginning of 2001. During the brainstorming phase, SAP had to consider a wide range of design criteria and combine them into a single toolset that was capable of meeting all its needs. The following criteria were foremost in its considerations:

1. Create a UI programming paradigm that would become the standard for all future SAP software. Ease of use is the key here.
2. Eliminate the repetitive coding tasks currently experienced by Web developers. The fewer lines of handwritten code there are in the UI, the better.
3. Make full use of abstract modeling. The application should not care about either the communication technology required to access a back-end system or the client technology used to render its screens.
4. Make full use of generic services. Functionality that is frequently required should be made available from a standard library of services. For example, if the user is asked to enter a date, then a date picker should automatically appear next to the input field without requiring extra coding effort on the part of the developer.
5. Use a declarative approach for application design. With this approach, the developer tells the development toolset *what* should be done, but not *how* to do it. This concept should extend into all areas of application design (e.g., screen flow and component reuse).
6. Change the focus of software reuse from low-level units of code to high-level units that represent distinct steps in a business process.

These design concepts have been followed very closely, and the result is a UI development environment strong enough in its functional capability and wide-ranging enough in its scope to be adopted by SAP for *all* its future application user interfaces.

Myth: "Web Dynpro has something to do with SAP NetWeaver Master Data Management (MDM) or SAP NetWeaver Exchange Infrastructure (XI)."

Fact: Web Dynpro, MDM, and XI are unrelated. Web Dynpro is a development toolset for building custom business applications. MDM and XI are technical systems in your SAP landscape, each with its own business purpose. MDM handles coordination, translation, and management of master data across multiple SAP systems. XI facilitates the exchange and translation of messages between different systems, the design and execution of business process workflows, and centralized system monitoring.

Note!

Web Dynpro applications run on an SAP NetWeaver Application Server, which underlies all of SAP's solutions, including MDM and XI.

Myth: "Web Dynpro doesn't require any programming."

Fact: Web Dynpro application development is not 100% coding free.[5] Although you can accomplish much of the development process using graphical tools (e.g., laying out UI elements on screens or describing navigation between screens), coding will always be required to implement the required business functionality. Where you don't need to write any coding is in the low-level areas such as the UI (for example, when creating HTML and JavaScript) or handling session management in the server or back-end communication.

Myth: "Web Dynpro programs must be written in Java."

Fact: With the release of SAP NetWeaver 2004s (and beyond), you can write Web Dynpro applications in either Java or ABAP.[6] The language you choose depends largely on the development resources you have available. Web Dynpro for ABAP applications run on any SAP NetWeaver 2004s or higher ABAP server, and Web Dynpro for Java applications run on any SAP NetWeaver '04 or higher Java server. Of course, if you have both the ABAP and Java stacks installed, you can use whichever flavor you prefer.

Myth: "Web Dynpro is the replacement for ABAP."

Fact: This is like the aircraft industry saying, "We're going to phase out the jet engine." The ABAP language is the mainstay of all SAP systems and there is no logical reason to remove it.[7]

Myth: "Web Dynpro applications are actually Java Server Pages in disguise."

Fact: Wrong! Not even close! Web Dynpro is based on SAP's own framework, which is not built upon anyone else's technology.

Myth: "Web Dynpro can only get data from SAP systems."

Fact: Web Dynpro applications can call Web services, ABAP function modules, or locally deployed Enterprise JavaBeans (EJB). In fact, there is no particular requirement even to have a back-end SAP system available.

Myth: "Web Dynpro is a design tool for building Web sites."

Fact: No, Web Dynpro allows developers to create business applications that form the content of a Web site, but this is not the same as creating the whole Web site. If you want to use SAP tools to create a Web site, then you should use SAP NetWeaver Portal, a sophisticated system for delivering and managing content in a highly structured, role-based environment.

> ***Note!***
>
> Technically speaking, Web Dynpro applications do not require the use of SAP NetWeaver Portal; however, the portal supplies many usability features such as role-based content delivery, single sign-on (SSO), object-based navigation (OBN), and portal eventing.

Myth: "Web Dynpro applications can run on any J2EE compliant server."

Fact: Although Web Dynpro applications are designed to be front-end and back-end neutral (i.e., you can access them from a wide variety of clients and obtain data from both SAP and non-SAP back-end systems), you can run the Web Dynpro applications themselves

[5] SAP has another development tool called Visual Composer and, as the name implies, development is 100% visual. Visual Composer allows you to build a basic business application without writing a single line of code by using a technique known as graphical programming. Web Dynpro, however, is not a graphical programming tool. For more information on Visual Composer, read Karl Kessler's article "Get started creating SAP Enterprise Portal iViews with Visual Composer — a purely model-driven, code-free development approach" (*SAP Professional Journal*, November/December 2005).

[6] Prior to SAP NetWeaver 2004s, Web Dynpro applications could only be written in Java.

[7] The average SAP system contains approximately 40 million lines of ABAP coding!

Will SAP port the Web Dynpro runtime to run on other J2EE servers?

This question is only relevant for people using Web Dynpro for Java; however, it is one that I have been asked several times.

Looking at the architecture shown in Figure 1 and Figure 2 on page 41, you may already be able to see why an SAP-supplied server is needed to run a Web Dynpro application: The executable application shown on the right sides of Figures 1 and 2 makes the Web Dynpro application appear to be some kind of self-contained, ready-to-run unit of code; however, this is not exactly the situation.

For the UI part of the application to be provided for you, all Web Dynpro applications must execute within a standard environment known as the Web Dynpro Framework (WDF). This ensures that the application maintains complete neutrality toward not only the client layer, but also the back-end system. So a deployable Web Dynpro application is a self-contained unit of code, but it is one that can only be executed by the WDF. This is why Web Dynpro applications may only be deployed to an SAP Java server.

The Web Dynpro developers have discussed at length the possibility of porting the WDF to another server platform. The main issue here is that it is not just the WDF that would have to be ported.

Within the SAP Java server, there are many units of functionality known as services — one of which is the WDF. Others include a service for handling Web services, a Java Dictionary service, all the Adobe Document services, and a Secure Store service, to name just a few of them. All of these services can be (and often are) used in conjunction with Web Dynpro. Therefore, to ensure true portability of a Web Dynpro application, not only must the WDF be ported, but also all the associated services. This effectively means porting the entire SAP Java server.

This proposal did not fare well in a cost/benefit analysis!

only on an SAP NetWeaver Application Server. See the sidebar above for more information.

Myth: "Web Dynpro is just another tool I'll add to my development arsenal."

Fact: There is nothing to stop you from continuing to use other Web development tools alongside Web Dynpro. However, as you learn more about Web Dynpro, you'll probably agree that Web Dynpro represents a quantum leap in designing Web-based interfaces to business applications. Web Dynpro lets you focus on the business objectives of your application and handles all the low-level processing for you. For most companies, whether using the ABAP or Java versions, Web Dynpro will become their tool of choice for building Web-based business applications.

Myth: "Developers can pick up Web Dynpro without extensive training."

Fact: Web Dynpro is not like anything SAP has ever done before. As stated previously, it represents a quantum leap in the design of Web applications because it provides a development framework in which the business application can genuinely be designed to be both client and back-end neutral. Therefore it is strongly recommended that developers attend the standard SAP training courses on Web Dynpro[8] to learn how to take full advantage of its available features.

[8] The codes for the "Introduction to Web Dynpro" training courses are NET310 for the ABAP version and JA310 for the Java version. For Web Dynpro for Java, there is also an advanced training course — its code is JA312.

How is Web Dynpro different from earlier UI technologies released by SAP?

The earlier UI technology used by SAP (the classical Dynpro technology used by SAPGUI) has its origins in the IBM frame-based communication protocol used for the old 3270 green screens. This protocol made no attempt to separate data presentation from data processing, and consequently, this lack of separation became a fundamental part of the SAPGUI protocol (known as the Dynamic Interactive Application Gateway, or DIAG).

When the World Wide Web boom started in the late 1990s, several things rapidly became clear:

1. The DIAG protocol used by SAPGUI was not suitable for these new things called "browsers."
2. Whatever technology was used would have to provide a clear separation between data presentation and data processing.

This was the motivation for SAP to go back to the drawing board and create a new UI development environment. By definition, it could not be based on anything SAP had done before. All the old SAPGUI concepts of Process Before Output (PBO), Process After Input (PAI), screen layout, and navigation had to go out the window and be replaced with what is now known as Web Dynpro.

With Web Dynpro, think *revolution*, not *evolution*.

I'll say it again: Do not think Web Dynpro is like any other development tool you've ever used before (see the sidebar above for more information). It has some similarities to other Web development toolsets, but on the whole, Web Dynpro requires a new way of thinking that, when grasped, will enable you to write efficient, low-maintenance business applications in less time than before.

Now that you understand both what Web Dynpro is and what it is not, let's explore the basic concepts you'll need to understand to successfully build Web Dynpro applications.

The Web Dynpro development environment

You can build Web Dynpro applications using one of two development tools: SAP NetWeaver Developer Studio[9] (NWDS) for Java, as shown in **Figure 1**, or the ABAP Development Workbench, familiar to all ABAP developers, as shown in **Figure 2**.

One of the best things about building Web Dynpro applications is that their architecture is defined declaratively, rather than programmatically. This means that you use graphical tools to build units of the application — for example, screen layout, the navigation paths from one screen to another, and the data structures used to hold the business data — and then the Web Dynpro development environment ties them together to create the final application.

Applications created just by declarations? Don't I have to write some code?

Yes, you do have to write some code, but using Web

[9] Development tool based on the open source tool Eclipse.

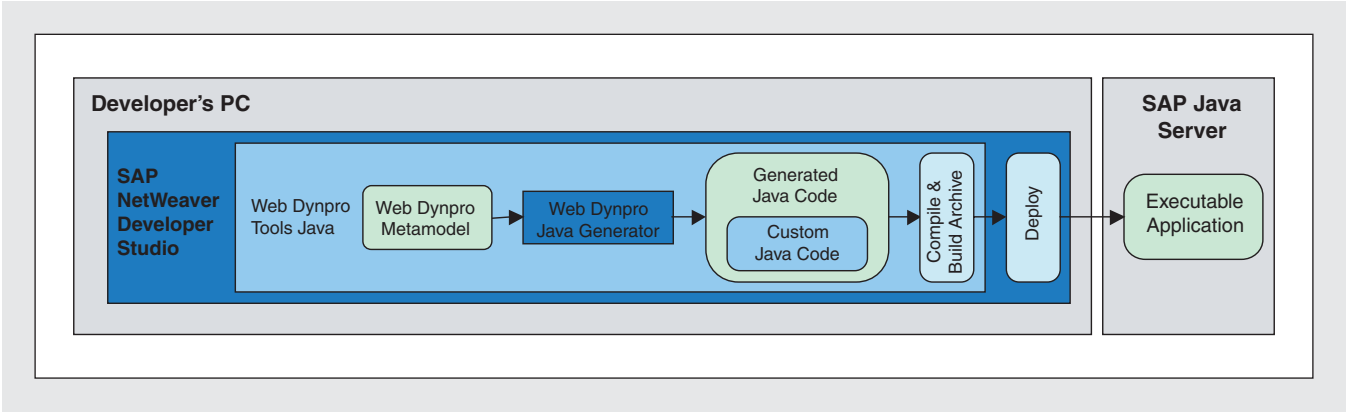

Figure 1 The Web Dynpro development process for Java

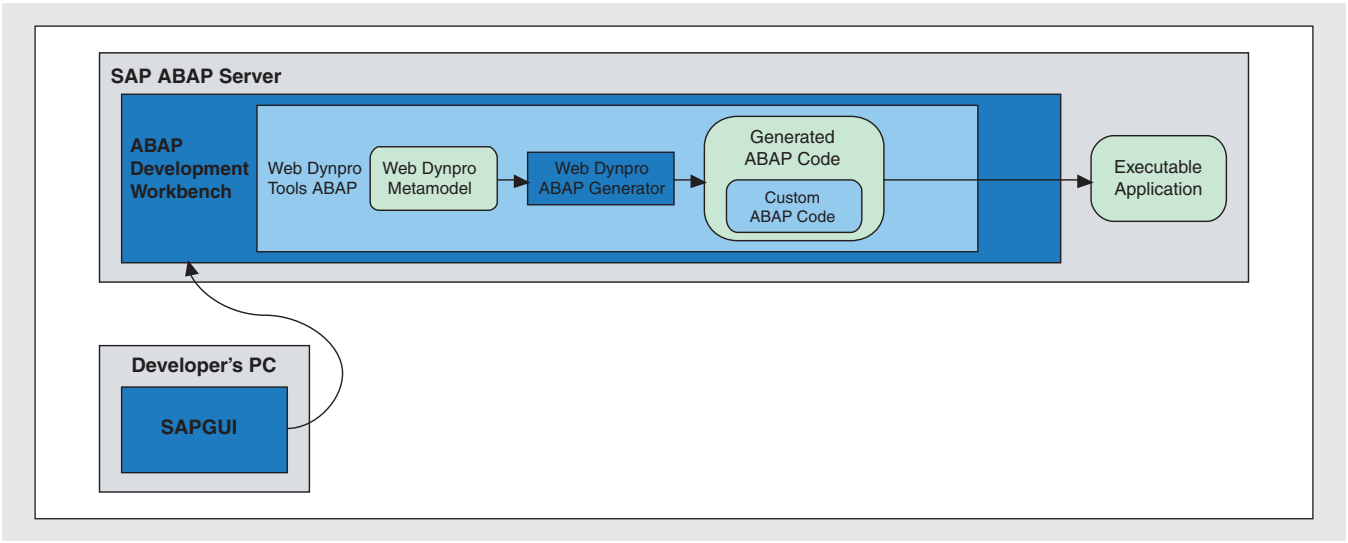

Figure 2 The Web Dynpro development process for ABAP

Dynpro, the quantity has been significantly reduced. The declarative part of the application design is where you specify:

- The major structural units of the application (known as components) and how they interact
- How information is to be displayed on the screen
- What data structures are to be used to hold the displayed data
- The various navigation paths from one screen to another
- How information is shared within the different components of an application
- Interaction with a business back-end system

Taken together, these declarations form a description of your business application known as the Web Dynpro "metamodel."

The original development environment idea

Originally, SAP intended to have a single, language-

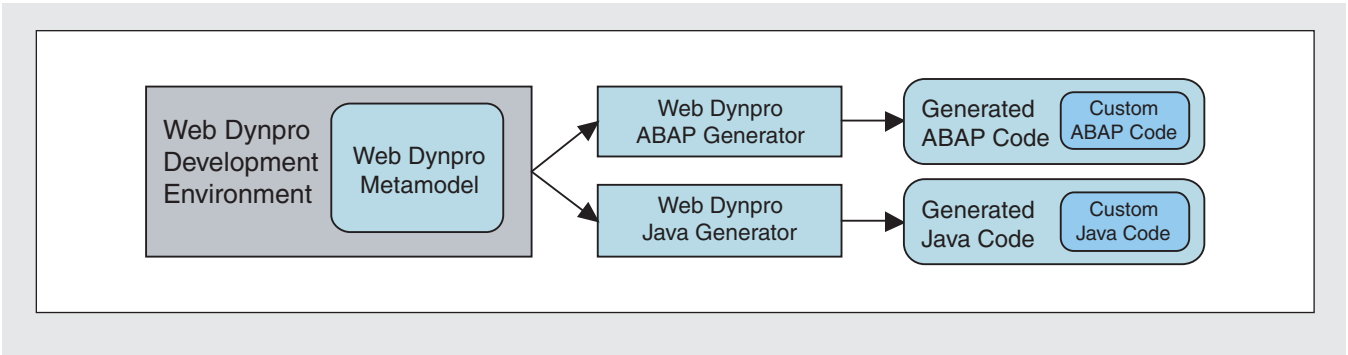

Figure 3 Original Web Dynpro design concept

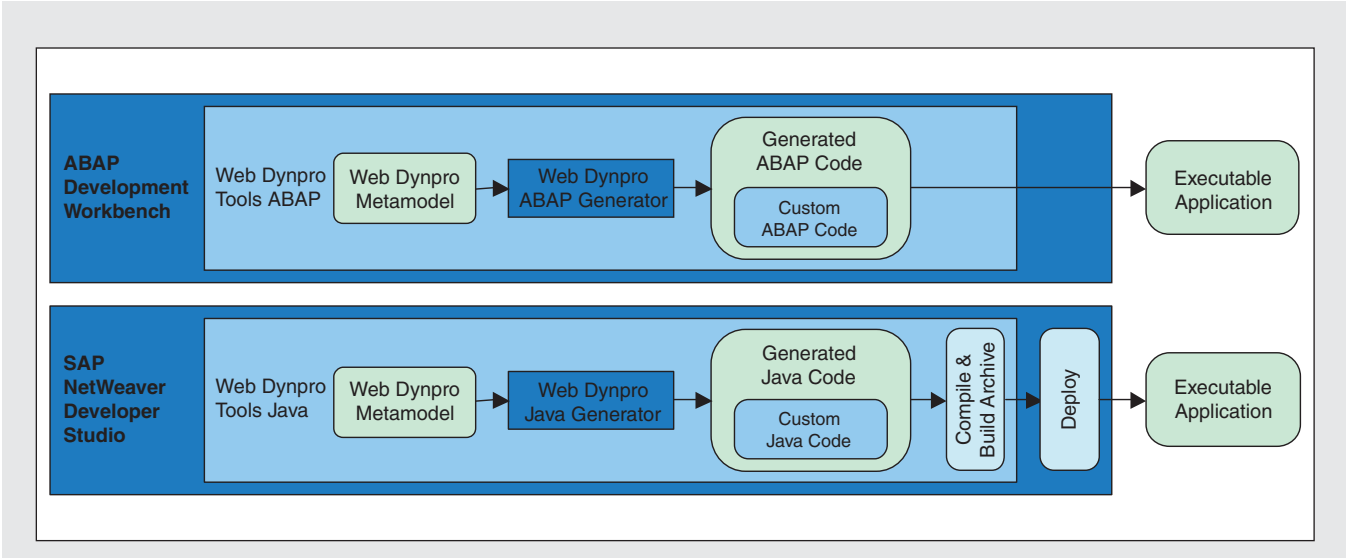

Figure 4 The actual Web Dynpro development environment

independent development tool in which the abstract metamodel would be created. Then the information from the metamodel would be fed into the appropriate Java or ABAP code generator to create the required coding. All you would then need to do is write the additional coding to manipulate the business data. See **Figure 3**.

The actual Web Dynpro development environment

As the development of Web Dynpro started, it rapidly became clear that it would be far more economical to have the Web Dynpro tools and code generator within the confines of a language-specific development environment. Consequently, all the tools for developing Web Dynpro for Java applications can be found within the Java development environment,[10] and all the tools for developing Web Dynpro for ABAP applications can be found within the ABAP Development Workbench.[11] Thus there are two independent Web

[10] The development of Web Dynpro programs in Java is performed in SAP NetWeaver Developer Studio (NWDS). This is the standard Eclipse development tool to which SAP has added a large number of extensions known as plug-ins.

[11] The ABAP Development Workbench (transaction SE80).

Dynpro development environments: one for Java and one for ABAP, as shown in **Figure 4**.

Getting started with Web Dynpro

There are two key concepts to know before developing Web Dynpro applications that will help you avoid major frustration:

- The Model View Controller (MVC) design concept (upon which Web Dynpro is based), and the unique way in which SAP has employed it
- The Web Dynpro component and its internal structure

While many of you may be familiar with the MVC design paradigm, SAP has modified and extended it, so you need to understand what the differences are, why they are there, and what consequences those differences will have on the way you think when writing Web Dynpro applications.

The Model View Controller (MVC) design concept

The MVC design concept is not at all new.[12] In fact, it was created in 1978 by a Norwegian software designer named Trygve Reenskaug.[13] At that time, he was working as a visiting scientist in the Smalltalk group at Xerox PARC and was faced with the problem of designing a system in which users would have to control large and complex datasets. Here's how he assessed part of the problem:

> ***Problem***
>
> The input and output aspects of the [program] are technically very different with few interdependencies. Their combination in a single object tends to make this object unnecessarily complex.
>
> ***Solution***
>
> Let the [program] contain two objects; a View object responsible for presentation, and a Controller object responsible for taking and interpreting input from the user.[14]

This is the very first time the idea of separating data presentation from data processing had actually been stated as an explicit design requirement. This idea is commonplace today, but in 1978 it was a massive conceptual leap.[15] The result is that he split the program up into at least two parts: the *view* and the *controller*, with each part having a very distinct role to play.

Having the view and controller was all very well, but the part of the program that would perform the actual business processing had still not been identified. Reenskaug recognized that a third unit of software was needed in order to have a fully functional program.[16] This is where the *model* came in. To use Reenskaug's own words again:

> ***MODELS***
>
> Models represent knowledge.
>
> A *model* could be a single object (rather uninteresting), or it could be some structure of objects… The nodes of a model should therefore represent an identifiable part of the problem.[17]

So, the *model* represents what the user knows (e.g., how to create a sales order), and all the information held within it relates to the business task at hand, or if the task is very complex, relates to some well-defined stage of the business task. The *view* is the part with which the user interacts (i.e., what appears

[12] For more information on building Web applications using the MVC design concept, see the *SAP Professional Journal* articles "Build More Powerful Web Applications in Less Time with BSP Extensions and the MVC Model" (March/April 2003) and "Develop More Extensible and Maintainable Web Applications with the Model-View-Controller (MVC) Design Pattern (January/February 2004).

[13] Pronounced "TRIG-vuh RAINS-cow."

[14] Reenskaug, Trygve, "The Model-View-Controller (MVC): Its Past and Present," 2003, Oslo, Norway: http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf.

[15] Nowadays, this is a routine design process known as "decoupling."

[16] Actually, Reenskaug identified a fourth unit of code known as an "editor," but this is not relevant to our current discussion.

[17] Reenskaug, Trygve, "MODELS-VIEWS-CONTROLLERS," 1979, Oslo, Norway: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf, page 1.

onscreen, such as menus, push buttons, drop-down lists, and option buttons). Finally, the *controller* is the clever part — this is where the processing takes place to determine the views the user sees on the screen, and then, based on the user interaction, determines the views the user sees next.

The essence of the MVC design concept

Many people think of MVC as a means for separating data presentation from data processing, and this is certainly true. However, the separation of data presentation from data processing is only one use case of a much more fundamental concept — namely, the separation of those parts of the program that generate data from those parts that consume data — and it is this fundamental principle that you see time and again within the Web Dynpro architecture.

Those parts of a Web Dynpro application that generate data (the models) are strictly separated from those parts of the application that consume data (the views). A controller acts as a middleman between the models and the views, and its role as a consumer or generator varies depending on which entity is being communicated with.

This principle has been applied in many different areas of Web Dynpro design and at different scales:

- **Model:** A model is *always* a data generator. Even though models receive data as input, that data serves only to drive some step of the business process, which consequently results in the generation of data.
- **View:** A view is *always* a consumer of data — whether raw data from the user via the keyboard and mouse, or processed data from a controller.
- **Controller:** A controller plays a dual role. When receiving data from a model, it acts as a consumer. However, after processing the model data, it passes it on to the view for display, and as far as the view is concerned, the controller acts as a generator.

When designing and writing Web Dynpro applications, you must be careful not to blur the boundaries between data consumers and data generators. If, through lack of understanding or wanting to achieve a quick result, you do blur these boundaries, then it is possible that you will end up writing poor quality, inefficient code.

I cannot stress enough the importance of understanding this concept because it is fundamental to the whole Web Dynpro component architecture, which we'll discuss shortly. Understanding the generator/consumer concept explains why SAP has extended Reenskaug's original concept and consequently implemented an MVC toolset that is different from other vendors' MVC implementations. Therefore, if you can adapt your way of thinking to understand SAP's implementation of the MVC paradigm, you will find understanding the Web Dynpro architecture very straightforward.[18]

Let's look at how SAP modified the MVC design concept.

SAP's adjustment to the view concept

To achieve the required level of client independence, SAP found it necessary to modify the concept of a view. Traditionally, views have been implemented as templates containing HTML and some executable coding; for example, Java Server Pages (JSPs) are HTML templates containing embedded Java coding. Unfortunately, this type of implementation carried with it all the problems SAP was trying to escape from in Web development. With views implemented as HTML templates, the developers needed to be concerned with such client-specific rendering issues as:

- HTML
- JavaScript
- Cascading Style Sheets (CSS)
- Support for multiple client types

[18] In the same way, understanding procedural programming essentially requires nothing more than an understanding of Böhm and Jacopini's completeness theorem (1966), which states that "Any program logic, no matter how complex, can be resolved by arranging statements into sequences, loops, and branches."

- Session management over the stateless HTTP protocol

These development areas alone can consume in excess of 50% of the implementation timescale. This is exactly the situation the SAP Web Dynpro designers wanted to avoid. Instead, a Web Dynpro view needed to hold the UI definition in a client-independent manner. Then at runtime, the WDF determines which HTML and JavaScript statements are needed to render the screen on any particular device. To achieve this, SAP needed to modify the concept of an MVC view. The result is that now there are in fact two general categories of *controller*:

- Controllers that do not have a visual interface (the traditional MVC controller)
- Controllers that do have a visual interface (SAP's "view controller")[19]

It is within a view controller that you define the layout of the screen, but to avoid having to write any HTML or JavaScript, you define the screen by making simple abstract declarations: "I want an input field here" or "I want a drop-down list over there." This approach means you don't waste time struggling with HTML and JavaScript because the task of generating HTML and JavaScript has been delegated to the WDF. At runtime, the WDF renders your view for whatever type of client device is currently running the application (and this could be something other than a browser, for example, a Blackberry or a Nokia Communicator).

The view controller has its own section of coding that prepares data for display and responds to user actions on the client. All this coding has been designed to operate without you ever needing to worry about how the specific UI elements should be rendered on the client. I won't go into any further details at the moment. All you need to remember is that Web Dynpro displays information on the screen using a special type of controller (one with a visual interface).

[19] There is also something called an "interface view controller," which I'll cover later in the article.

So when I write a Web Dynpro program, I'm just writing views, models, and controllers?

Well, not exactly…

MVC is not a patented methodology and no one holds the copyright on it. MVC is just a design concept in the same way that object-orientation is a design concept. Therefore, every software vendor that implements an MVC-based design tool does so in a slightly different way and with a slightly different interpretation of the principles. SAP is no exception here. In fact, SAP has extended Reenskaug's original MVC concept by introducing a completely new entity known as a "component." Without this new entity, the Web Dynpro designers found they could not correctly implement one of their key design requirements: namely, code reuse at a business level.

The Web Dynpro component concept

When you write a Web Dynpro program, you *will* need to write models, views, and controllers. However, these units of code are aggregated together into a larger unit called a *Web Dynpro component*.[20]

When you write a Web Dynpro application, you must write at least one component. The component is now both your unit of development and your unit of reuse. (For more information, see the sidebar on the next page.)

When it comes to code reusability, MVC uses the following unspoken principle:

> Coding is bad…if you have to write the same piece of code twice.

Using Web Dynpro's component reuse concept, you should be able (with some careful design) to slash UI development times by more than 50% by creating a library of reusable components — each one corresponding to a distinct step of a business process.

[20] The Web Dynpro component is an SAP extension to MVC that is not part of Reenskaug's original specification.

Are Web Dynpro components really necessary?

It could be argued that *views*, *controllers*, and *models* could all be reused as independent units. This is true, but this was not the type of reuse SAP was after. If the only reusable units of code were low-level ones such as views and controllers, then you would create the situation in which object reuse was not directly related to any step of a business process.

Since the whole thrust of Web Dynpro was aimed at modeling and then solving *business* problems, the technical details of how the problem should be solved were of lesser importance. Therefore, the unit of software reuse should correspond to a distinct step of the business process, not some low-level, technical unit of coding.

Consequently, Web Dynpro developers no longer need to spend hours fiddling around with HTML and JavaScript trying to get some screen widget working in three different browser flavors. Instead, their focus is now on the flow of information through the business process — which is where it should be anyway.

When developers are now asked to write a new application, their job consists of:

1. Analyzing the business process to identify where currently available Web Dynpro components can be reused
2. Designing and writing new Web Dynpro components to fill any functional gaps
3. Assembling the components together into the required business application

The whole emphasis of code reuse in Web Dynpro focuses on discrete steps in the business process and not on the low-level units of code required to implement those steps. This means that a Web Dynpro component contains several distinct coding entities that all function together to deliver a single unit of business processing. A Web Dynpro application is then built using one or more Web Dynpro components as reusable building blocks.

The structure of a Web Dynpro application

The basic Web Dynpro application that we'll look at is a portion of the flight model from the SAP IDES training system. We will look at how the user can search for flights between certain airports and then select a flight to see its details. Obtaining a list of flights and then displaying the details of a specific flight are possible because the Web Dynpro application makes calls to the back-end SAP system. We'll break up the application into its constituent parts and examine its screens, the flow of information, and the reusable components from which it is built.

Note!

Strictly speaking, the simple application that we're about to look at does not need to be modularized to this extent, but the point here is to illustrate the principles upon which you can build more complex Web Dynpro applications — without actually having to build a complex application!

Let's begin with the screen the user encounters when trying to book a flight. As shown in **Figure 5**,

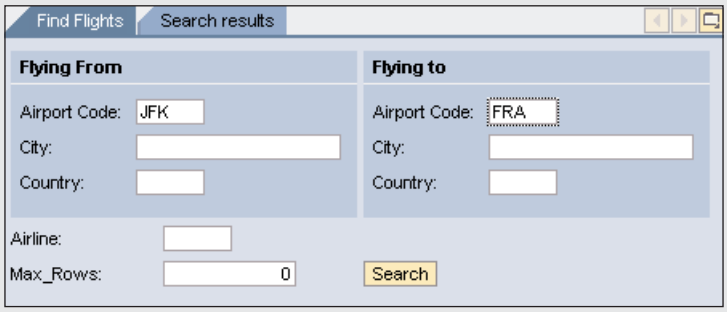

Figure 5 The first screen of the flight model application for entering search criteria

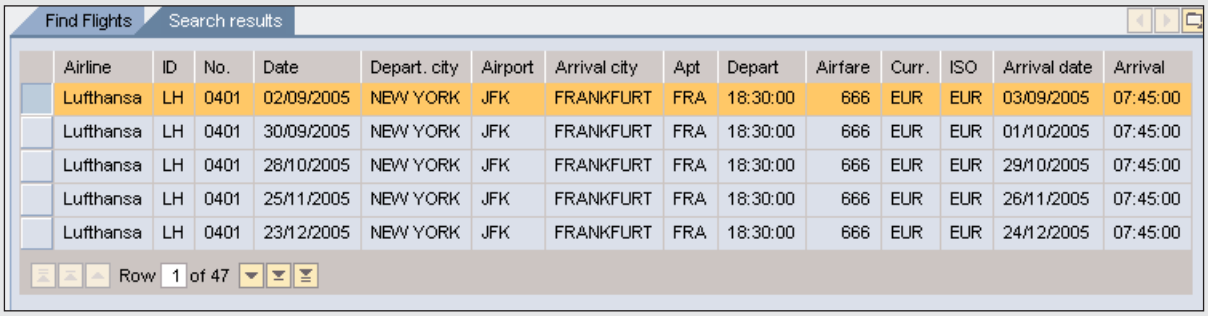

Figure 6 Search results for flights between New York's JFK and Frankfurt, Germany

the user enters some search criteria — that is, the airport codes to search for flights between New York's JFK airport and Frankfurt in Germany — in the Find Flights tab.

The user then clicks on the Search button, and the results are displayed in the Search results tab, as shown in **Figure 6**. You can see in the status line at the bottom of the table that 47 flights have been found.

Next the user scrolls down to a flight of interest. To display the flight details, the user simply clicks on the button to the left of the table row, and the information is displayed in a new screen area underneath. See **Figure 7** on the next page.

The application's design

Let's look at how the `FlightInformation` application has been constructed. Knowing that a Web Dynpro component is both the unit of development and the unit of reuse, you must first examine your business process to identify reusable units of code. You have:

- A search screen that receives flight information and returns a list of matching flights from the back-end system. This is the `FlightSearch` component.
- A details screen that receives information about a single flight and retrieves its details from the back-end system. This is the `FlightDetails` component.

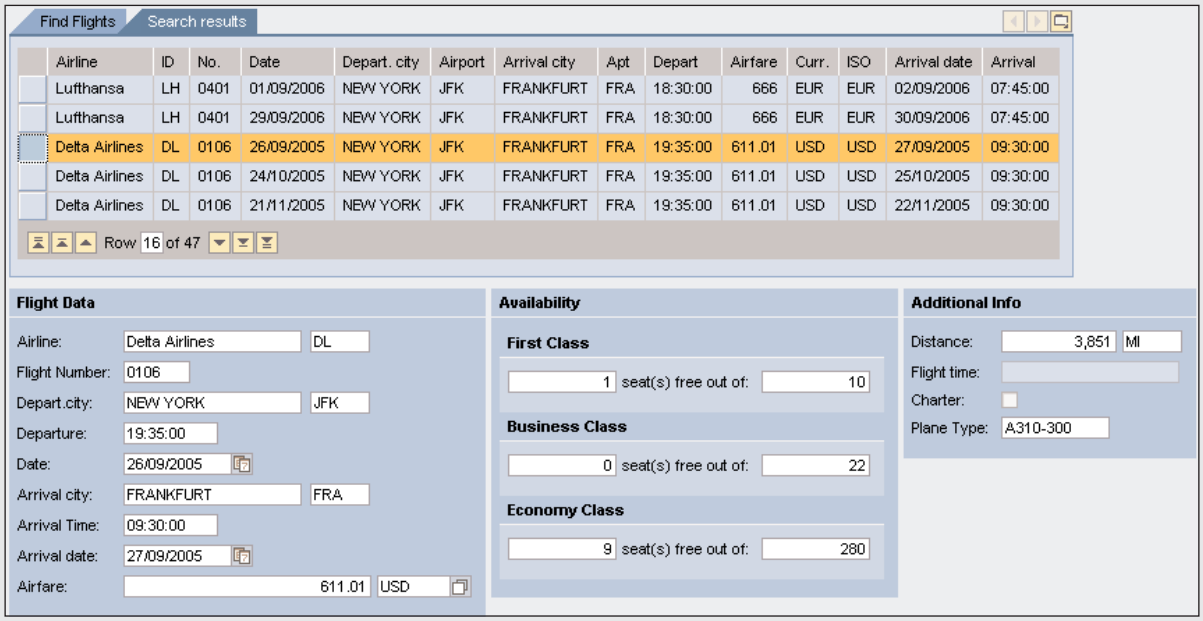

Figure 7 Flight details for Delta flight 106

You can implement these two units of functionality independently, since both searching for flight information and displaying flight details are generic tasks that could be performed in a variety of applications. To implement these two independent units of functionality, you need to write two components — one for searching for flights and one for displaying flight details — both of which are completely reusable.[21] You also need some way to glue these two components together. Therefore, you need a third component (called `FlightInfo`) to act as a container, or parent, for the `FlightSearch` and `FlightDetails` components. In this situation, this parent component does not need to be reusable (although technically, it is); rather, it represents the entire delivered business process. So looking at **Figure 8**, you can see that each component will be responsible for a different area of the screen.

The `FlightSearch` component manages the upper area of the screen, and the `FlightDetails` component manages the lower area of the screen. The `FlightInfo` component then joins these two components together so that they function as a pair. Notice that the parent component does not put any information on the screen itself. The `FlightInfo` component is simply a container that knows how to make the `FlightSearch` and `FlightDetails` components function together as a coherent business application.

Next, we will look at how the `FlightSearch` component operates. But before I explain how this component has been built, you must first have a look at how a Web Dynpro component functions in general. If you understand the general principles of a Web Dynpro component, then you will have no problem following how the `FlightSearch` component works.

Inside a Web Dynpro component

If you take the lid off a Web Dynpro component, you will see various units of code that are arranged in the way shown in **Figure 9** (in Web Dynpro for Java at least).

[21] I must reiterate at this point that an application this simple does not need to be implemented using multiple components, but the object of the exercise is to demonstrate the principles of good Web Dynpro design. Therefore, I need to demonstrate component reuse.

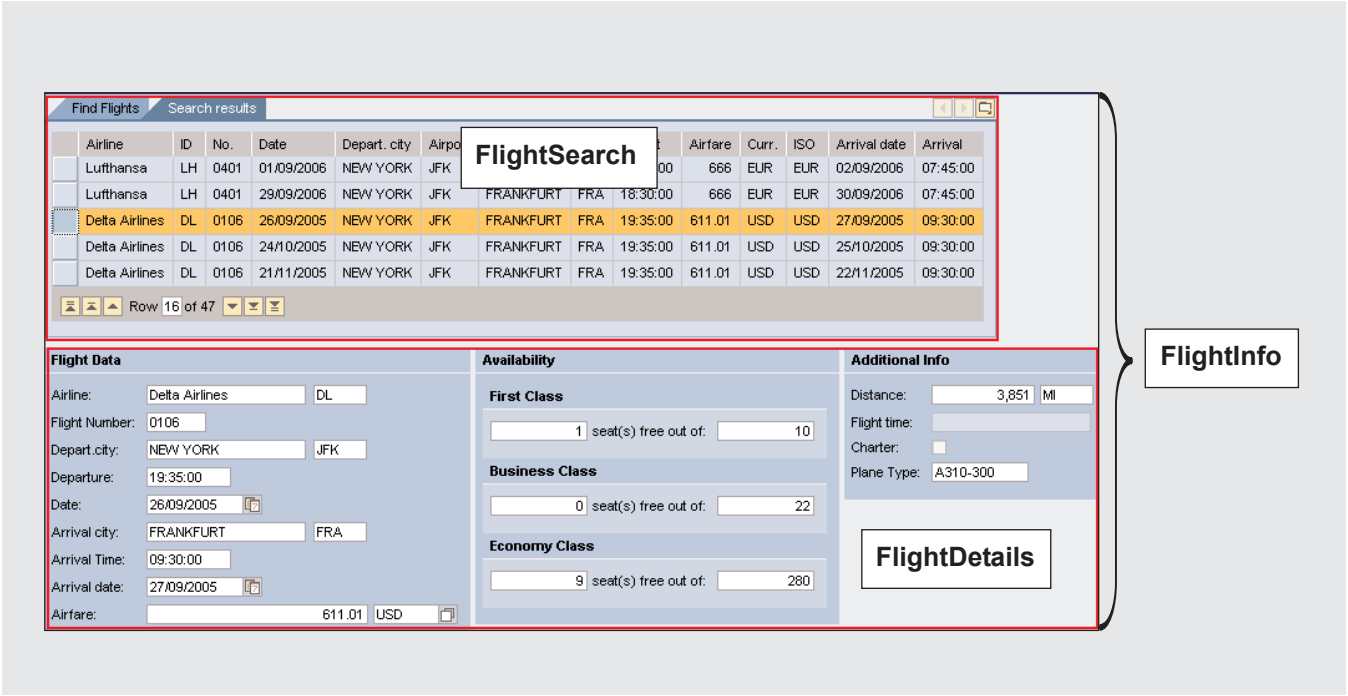

Figure 8 Different components supply information to different parts of the screen

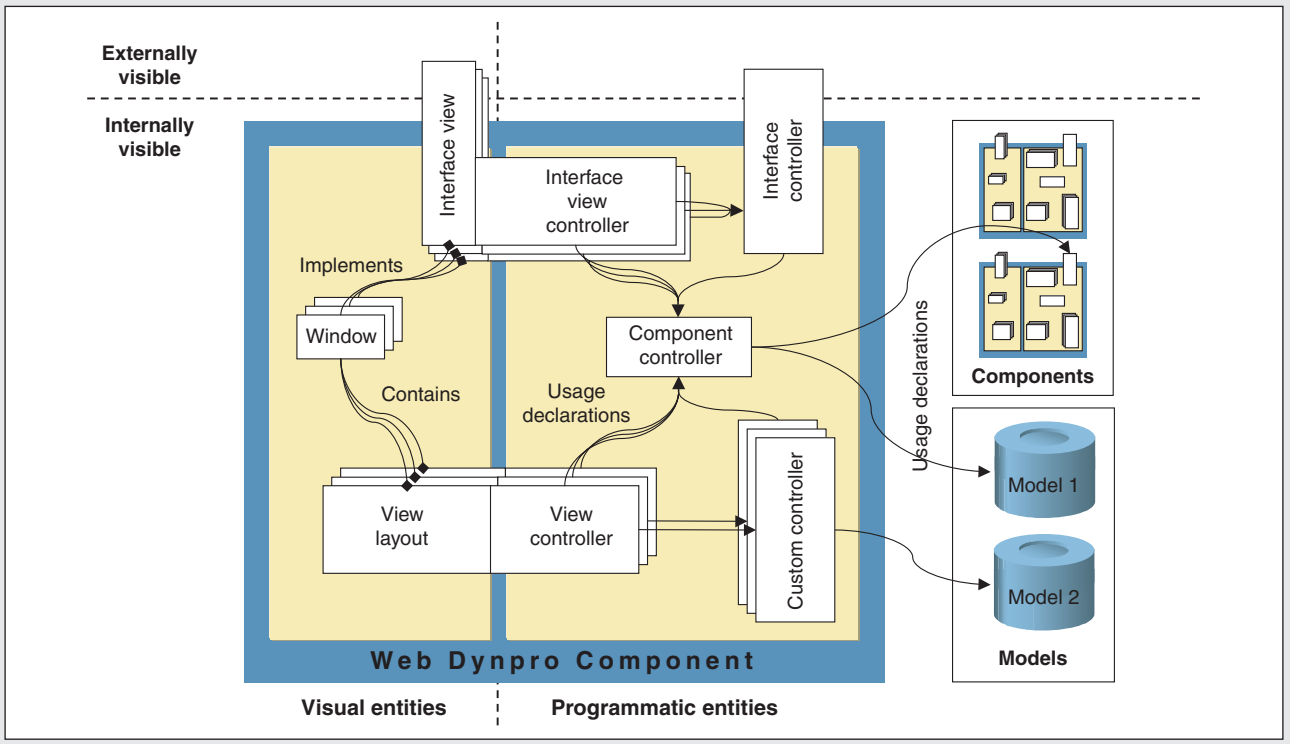

Figure 9 The structure of a Web Dynpro for Java component

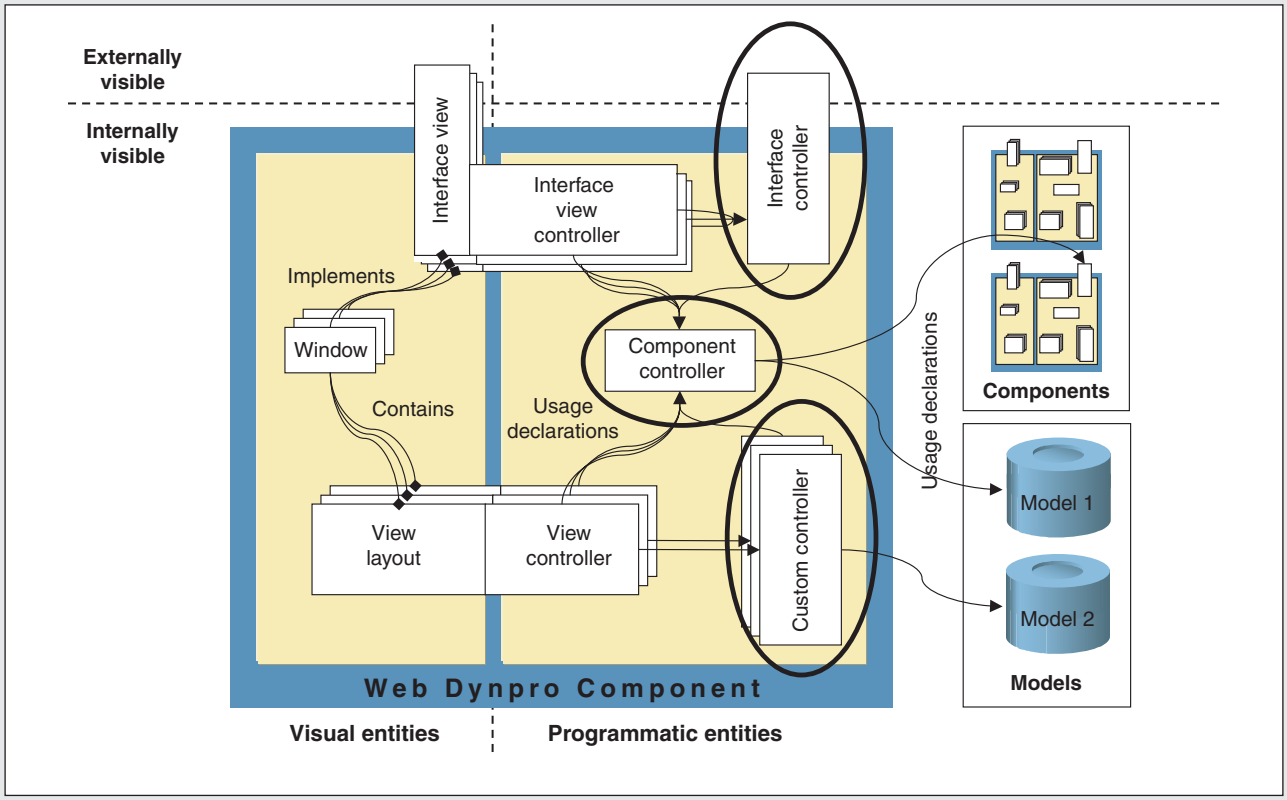

Figure 10 Controllers that have no visual interface

Note!

Web Dynpro for Java was first available to customers in early 2004 as part of the SAP NetWeaver '04 (NW04) release. However, Web Dynpro for ABAP was not ready for release until November 2005 as part of the SAP NetWeaver 2004s (NW04s) release.

Various improvements were made to the architecture of the Web Dynpro component between these releases. But since the NW04 version of Web Dynpro for Java had already been released to customers, modifying the architecture of the Web Dynpro component for the NW04s release would have created a significant impact for the customer's own developments. Therefore, the implementation of the new component architecture for Web Dynpro for Java has been delayed until the SAP NetWeaver 2007 release, which is scheduled to enter ramp-up sometime in the first half of 2007.[22]

Web Dynpro for ABAP, however, was in the advantageous position that there were no prior development projects using it; therefore, the component architecture it implemented could be the new one. Consequently, the component structure in Web Dynpro for ABAP is slightly different from the component structure in Web Dynpro for Java.

[22] This statement is correct at the time of printing. The release date of SAP NetWeaver 2007 may be modified, delayed, or canceled without prior warning or notice.

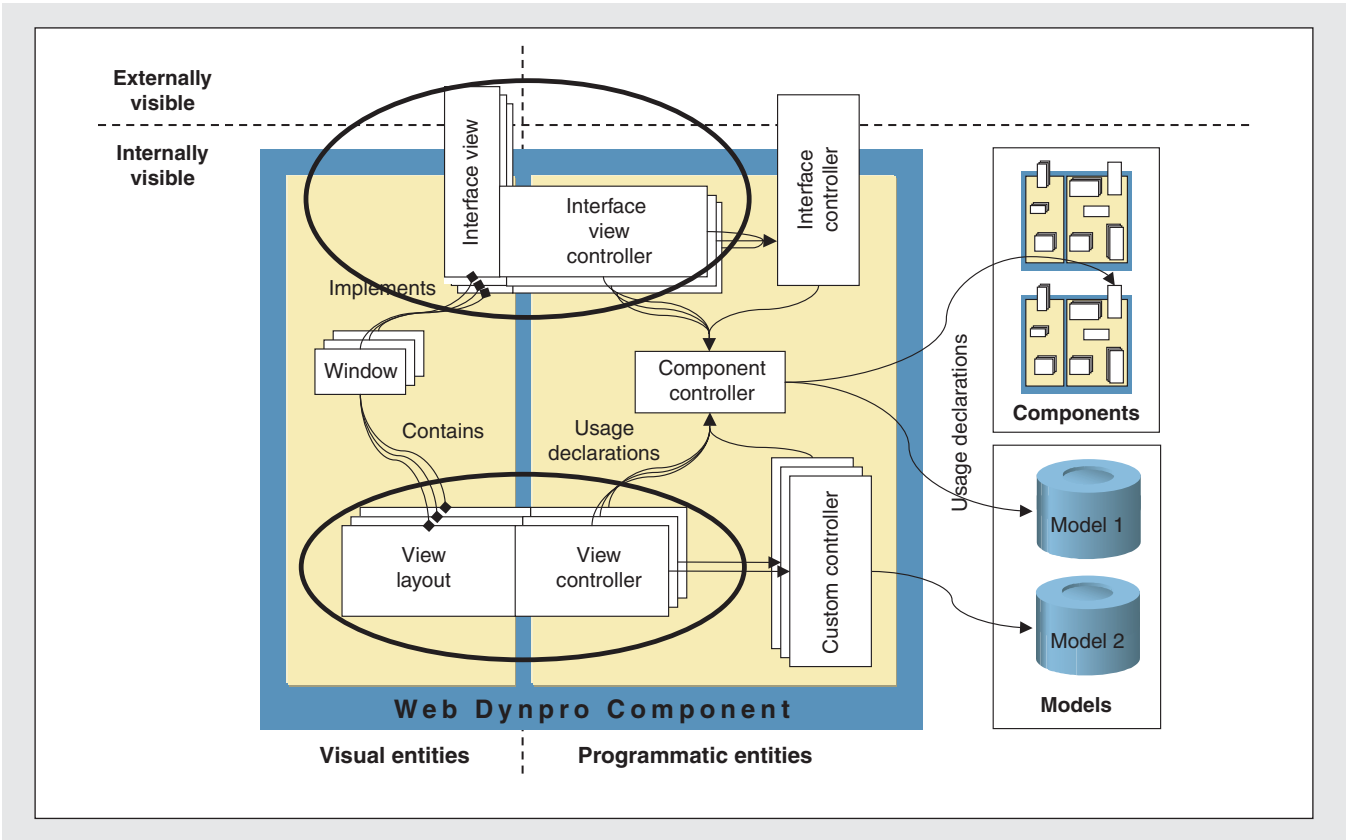

Figure 11 Controllers that do have a visual interface

High-level subdivisions

At the highest level, **Figure 9** shows four subdivisions within a Web Dynpro component: The vertical dotted line separates the *programmatic entities* on the right from the *visual entities* on the left. The horizontal dotted line separates the *internal (private) entities* below the line from the *external (public) entities* above the line.

On the right side, there are two blocks labeled Models and Components. These blocks indicate that a Web Dynpro component can make reference to any functionality found within models and other Web Dynpro components.

Two principles are at work here:

- **Code reuse:** Models are always defined outside the scope of a component and can therefore be reused by multiple components.
- **Generator/consumer separation:** The model always acts as a generator of data, and here the entire component is behaving as the consumer.

Where are the controllers that have no visual interface?

The Web Dynpro controllers that have *no* visual interface are identified in **Figure 10**. Although three different controllers are identified as having no visual interface, all three are known generically as *custom controllers*.

Where are the controllers that do have a visual interface?

The Web Dynpro controllers that *do* have a visual interface are identified in **Figure 11**. These controllers

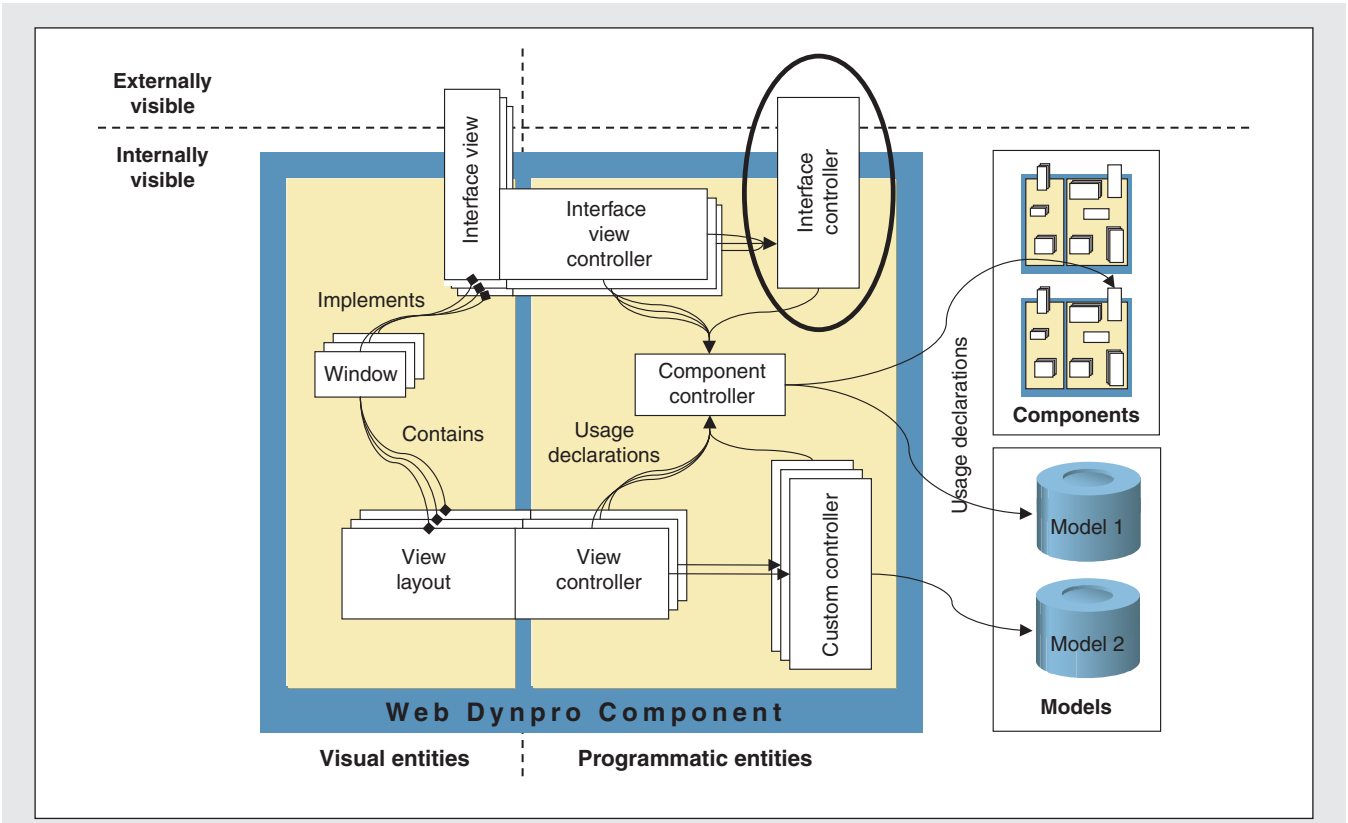

Figure 12 The programmatic interface to a Web Dynpro component

are responsible for user interaction through the client device.[23] Remember that these controllers are not responsible for the generation of data.

> ***Note!***
>
> A view controller is not responsible for generating the data it displays — that is the role of a custom controller. Remember, you must be careful not to blur the boundaries between the roles of a data generator and a data consumer. The reason that I will keep reminding you of this is because, technically speaking, it is possible to break the rules.

[23] Examples are a browser, a Pocket PC, or a Blackberry.

It is important to realize that the controllers *with* a visual interface play a very different role than the controllers *without* a visual interface. This is particularly important when writing the code to access a back-end system!

OK, what else do I need to know about components?

There is only one other thing you need to know about a Web Dynpro component. If you are going to spend all this time building clever bits of reusable functionality, how do you get to that functionality when you want to reuse it?

Here's where you need to look at the interface to a Web Dynpro component. In fact, a Web Dynpro component has two interfaces: one is programmatic (**Figure 12**) and the other is visual (**Figure 13**).

So when you want to reuse the functionality found in a Web Dynpro component, first ask yourself, "Does

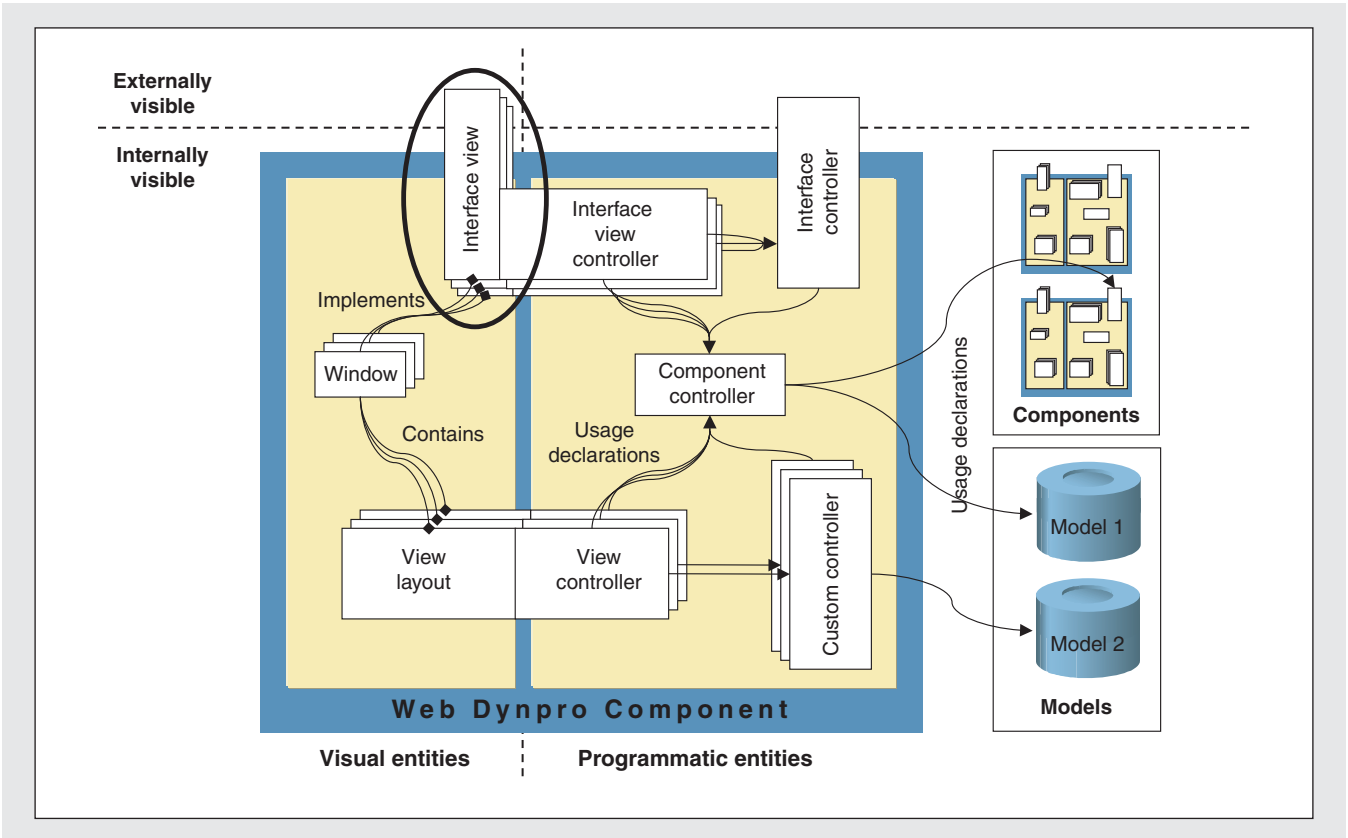

Figure 13 The visual interface to a Web Dynpro component

the component I am reusing need to put any data on the screen?" If the answer is "Yes," then you need to make use of the component's visual interface.

Once you have built the views for a component, you can reuse its entire visual interface as if it were a single UI element. On the other hand, if the component being reused does not put any data on the screen, then you only need to reference the component's programmatic interface.

Now that you have an overview of the internal architecture of a Web Dynpro component, let's go back and look at actually building the `FlightInformation` application.

Building the FlightSearch component

If we start with the visual interface, we can see two screens arranged in a tab strip. The first appears in the Find Flights tab (**Figure 5**). As you saw earlier, this is the area in which the user enters the search criteria. The second tab is Search results (**Figure 6**), which is where the search results are displayed. The overall screen structure, however, is built using three views. The two views (i.e., the Find Flights tab and the Search results tab) are embedded into a parent view in which the tab strip itself is defined, as shown in **Figure 14**.

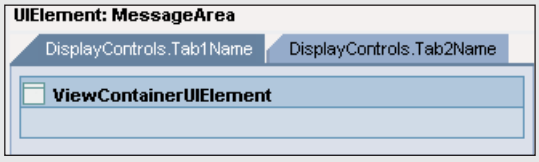

Figure 14 Parent view used to define the tab strip

Figure 15 Design time appearance of the Find Flights tab

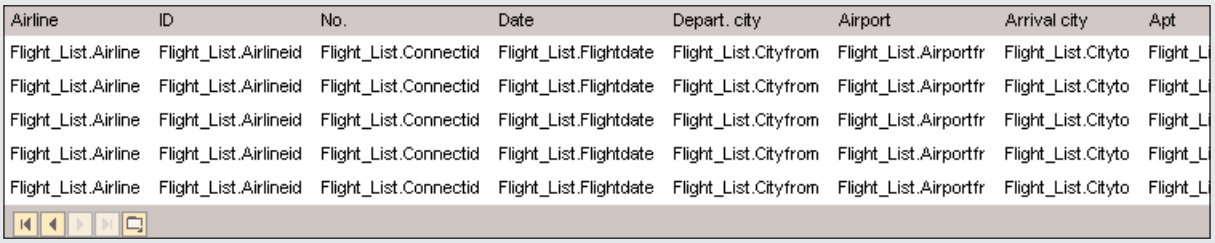

Figure 16 Design time appearance of the Search results tab

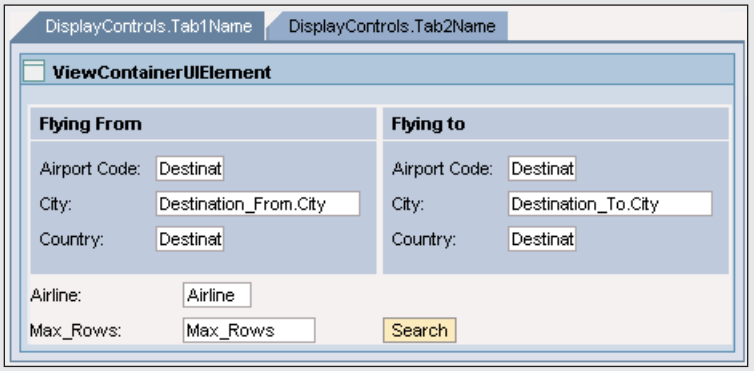

Figure 17 The search view embedded in the first tab

Figure 14 shows a UI element known as a `ViewContainerUIElement` embedded within the first tab (also in the second tab, but you cannot see it in the figure). This is the UI element that allows Web Dynpro to take the entire set of UI elements from another view or component and embed them into the current view as if they were a single UI element. **Figure 15** shows the view layout for entering flight details. This entire view will be embedded into the `ViewContainerUIElement`.

Notice that each of the input fields holds some text. This is not the text that appears on the screen at runtime; instead, it is the name of the field and data structure from which the UI element obtains its data at runtime.[24]

[24] For simplicity, I have not used the correct terminology here. In true Web Dynpro speak, I should say, "instead, it is the name of the attribute and context node to which the UI element property has been bound." However, explaining the meaning of these terms goes beyond the scope of this introductory article. Suffice it to say, the explanation I have provided gives you a correct understanding of the concept.

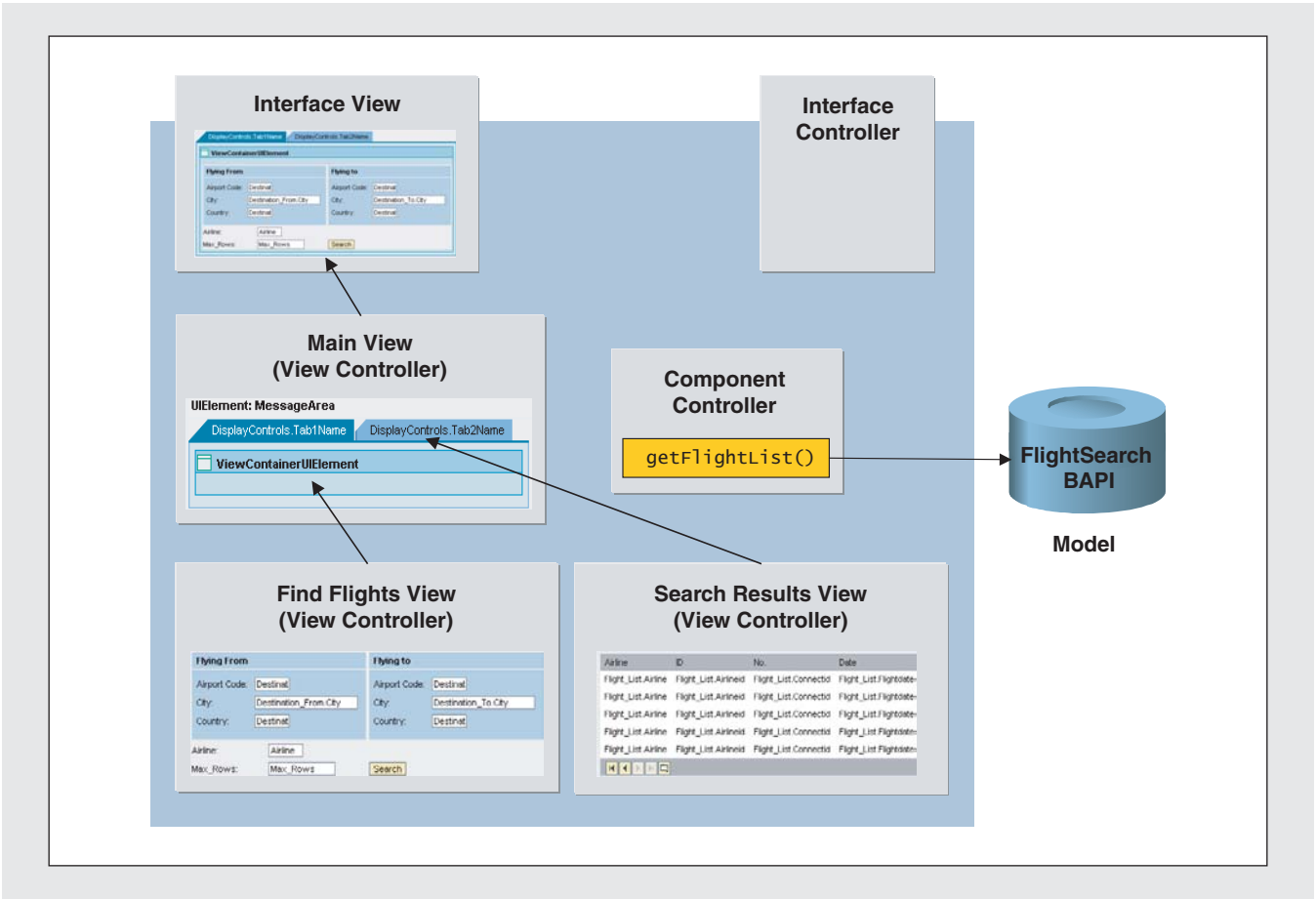

Figure 18 Block diagram of the FlightSearch component

When the user clicks on the Search button, the component sends the search criteria to a back-end system for a database query.[25] This is done by means of a Web Dynpro model. All Web Dynpro models represent the interface to some back-end service — such as a BAPI call or a Web service. The coding required to communicate with the back-end system is also embedded within the model.

Once the model that interacts with the back-end SAP system has generated the search results, the second tab is brought to the foreground and the search results are displayed. The same type of UI element used in the first tab has been used here; that is, the tab contains a single `ViewContainerUIElement` into which the view for displaying the search results has been embedded. See **Figure 16**.

So the overall visual interface of the `FlightSearch` component can be represented as shown in **Figure 17**.[26]

The same principle holds true for embedding the search results into the second tab. Here is a situation in which, within the scope of a single component, two views have been embedded into a third view to create the visual interface. **Figure 18** shows the overall structure of the `FlightSearch` component.[27]

[25] In the example, this is done by calling a BAPI in a back-end SAP system. Access to the BAPI is provided by a Web Dynpro model.

[26] This screenshot does not represent an actual screen from either SAP NetWeaver Developer Studio or the ABAP Development Workbench. Instead, it illustrates the principle that one view can be embedded into another.

[27] This diagram has been simplified and is only intended to represent the main conceptual units of the component. In reality, the component contains other entities not shown in this diagram. However, the important point here is the communication of concepts, not the precise technical details.

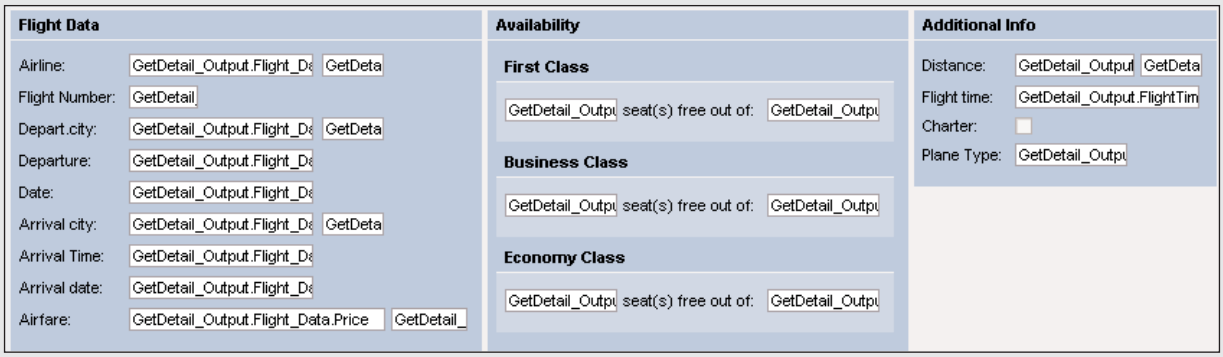

Figure 19 Design time appearance of the FlightDetails view

The interface view of a component contains all the views that we want to put together on the screen. Later on, you'll see how the interface view of this component can be picked up and reused by a parent component as if it were a single UI element.

Within the component controller, there is a method called `getFlightList()`. This is the part of the program responsible for taking the user's search criteria and passing them to the back-end SAP system via the model object. Once the search results are obtained, they are displayed in the Search results view.

Building the FlightDetails component

Displaying the details of a single flight has also been identified as a reusable task. Therefore, it too will be implemented as a single Web Dynpro component. In this situation, the `FlightDetails` component does not care who supplies it with data. All it cares about is that it receives enough information to uniquely identify a single flight. It uses this data and calls another ABAP function module in the back-end system to retrieve the information about one specific flight.

The `FlightDetails` component is simpler than the `FlightSearch` component because it only has one screen and displays data for output only (so we don't have to worry about reacting to any user input). Within the `FlightDetails` component, we must construct a single view that looks like **Figure 19**.

Designing the screen is all very well, but how does the `FlightDetails` component receive the information about which flight to read from the back-end system?

This is where the interface controller of the Web Dynpro component is required. Looking at **Figure 20**, you can see that within the interface controller there is a method called `showFlightDetails()`. Since this method exists in the interface controller, it is publicly visible and can be called from another Web Dynpro component. Once called, the `showFlightDetails()` method in the interface controller calls a method of exactly the same name in the component controller, which in turn calls the BAPI in the back-end SAP system via the model.

Once the BAPI has returned the relevant information, it is passed to the `FlightDetails` view controller for display. As with the `FlightSearch` component, the interface view of the `FlightDetails` component contains the sum total of all views to be displayed by this component, which in this case is only one.

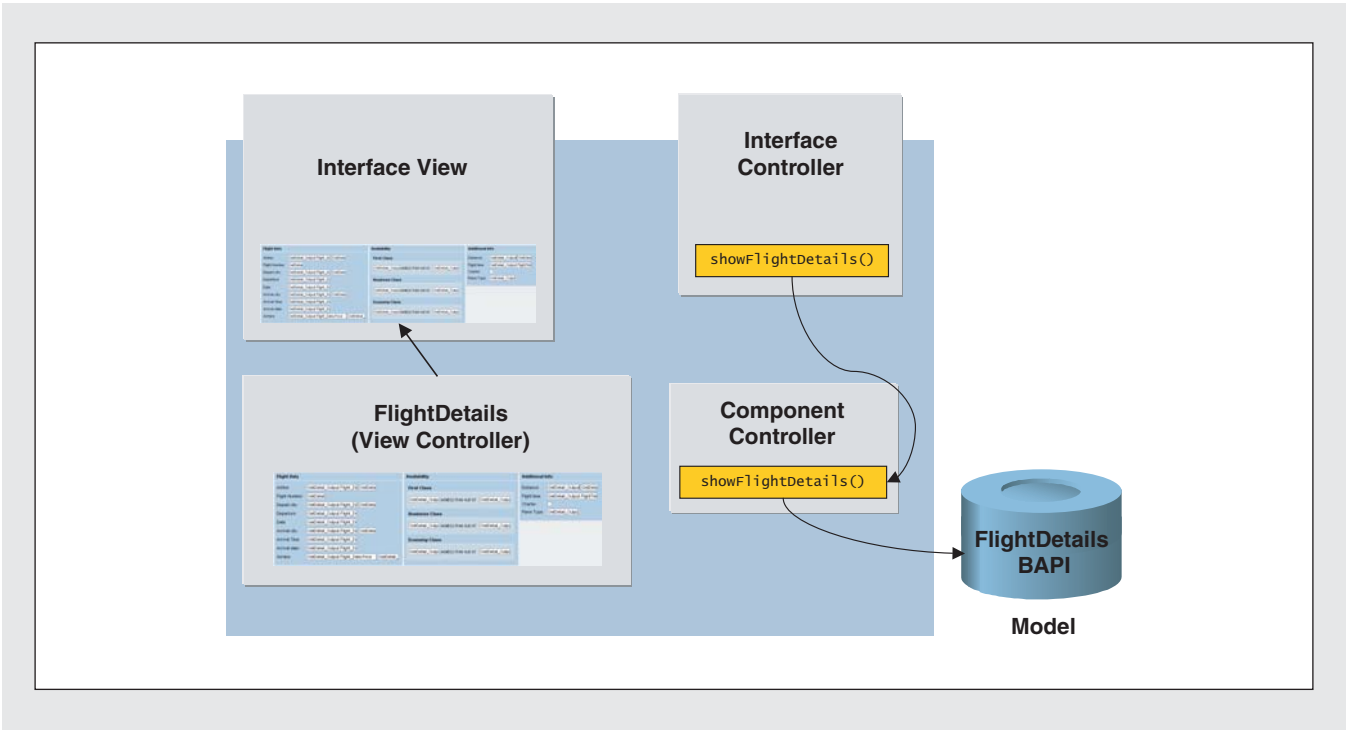

Figure 20 Block diagram of the FlightDetails component

Building the FlightInfo component

Now that we have our two reusable components, we must make them function together as a single business application. This is where we need the third and final component, `FlightInfo`. The purpose of this component is to act as a manager for the other two components. The `FlightInfo` component itself does not contain any business logic. Instead it contains the logic that allows the `FlightSearch` and `FlightDetails` components to interact with each other. **Figure 21** on the next page shows the structure of the `FlightInfo` component.

Notice that the `FlightInfo` component only has a single view controller called `FlightInfoView`. Remember I said that the entire interface view of a component can be picked up and reused as if it were a single UI element? Well, here's where that happens.

The interface views of the `FlightSearch` and `FlightDetails` components have been embedded into their own containers within the `FlightInfoView` view controller. So when the user selects a row from the flight results table, an event is raised to notify the parent component that a selection has taken place. The parent component then responds to this event and calls the `showFlightDetails()` method in the interface controller of the `FlightDetails` component.

Generating a URL to launch the application

At the moment, we have constructed a fully functional Web Dynpro application; however, we have not yet given the user any way of executing it. Therefore, the final step in the process is to generate a URL that points to the interface view of the `FlightInfo` component. This becomes the entry point into the Web Dynpro functionality we have just written.

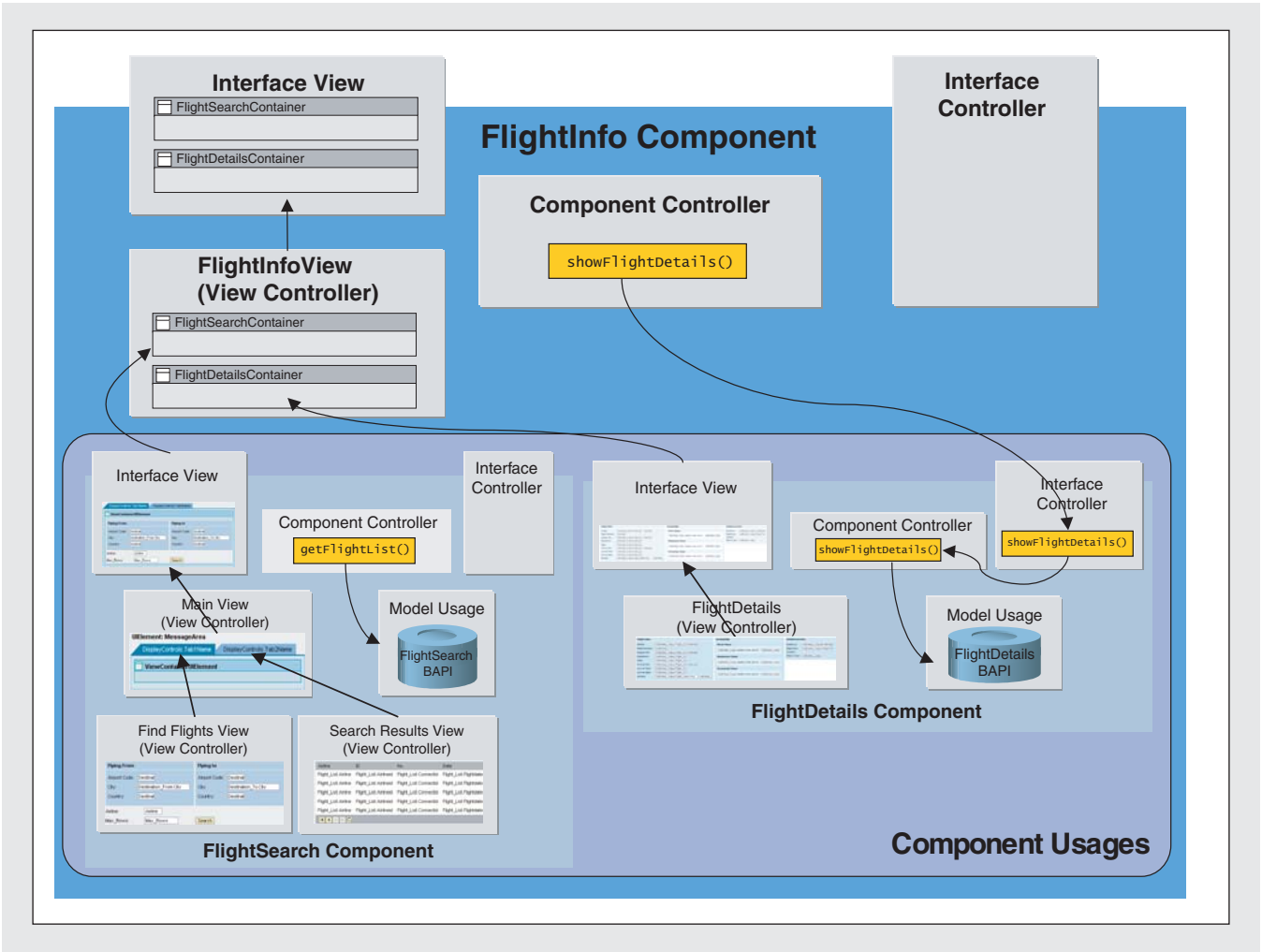

Figure 21 The FlightSearch and FlightDetails components embedded into the FlightInfo component (parent)

The URL that points to a Web Dynpro component is known as an "application," and is shown in **Figure 22**.

In order for users to be able to run the `FlightInfo` application from their browser, they simply need to enter the generated URL into the browser's address line. Users will not be expected to type in this URL since it is usually made available as a Web Dynpro iView in the SAP NetWeaver Portal.

Conclusion

This introduction to Web Dynpro has shown how you can build a simple flight information application. Along the way, you have had a history lesson that shows how Web Dynpro came into existence and how SAP has modified and extended the MVC design paradigm to produce a robust, enterprise-ready, Web development toolset.

Let's quickly recap what you've learned about Web Dynpro:

- Web Dynpro is a toolset for developing the UI layer of a business process. It has not been designed as a tool for creating an entire Web site.
- The Web Dynpro application does not care what

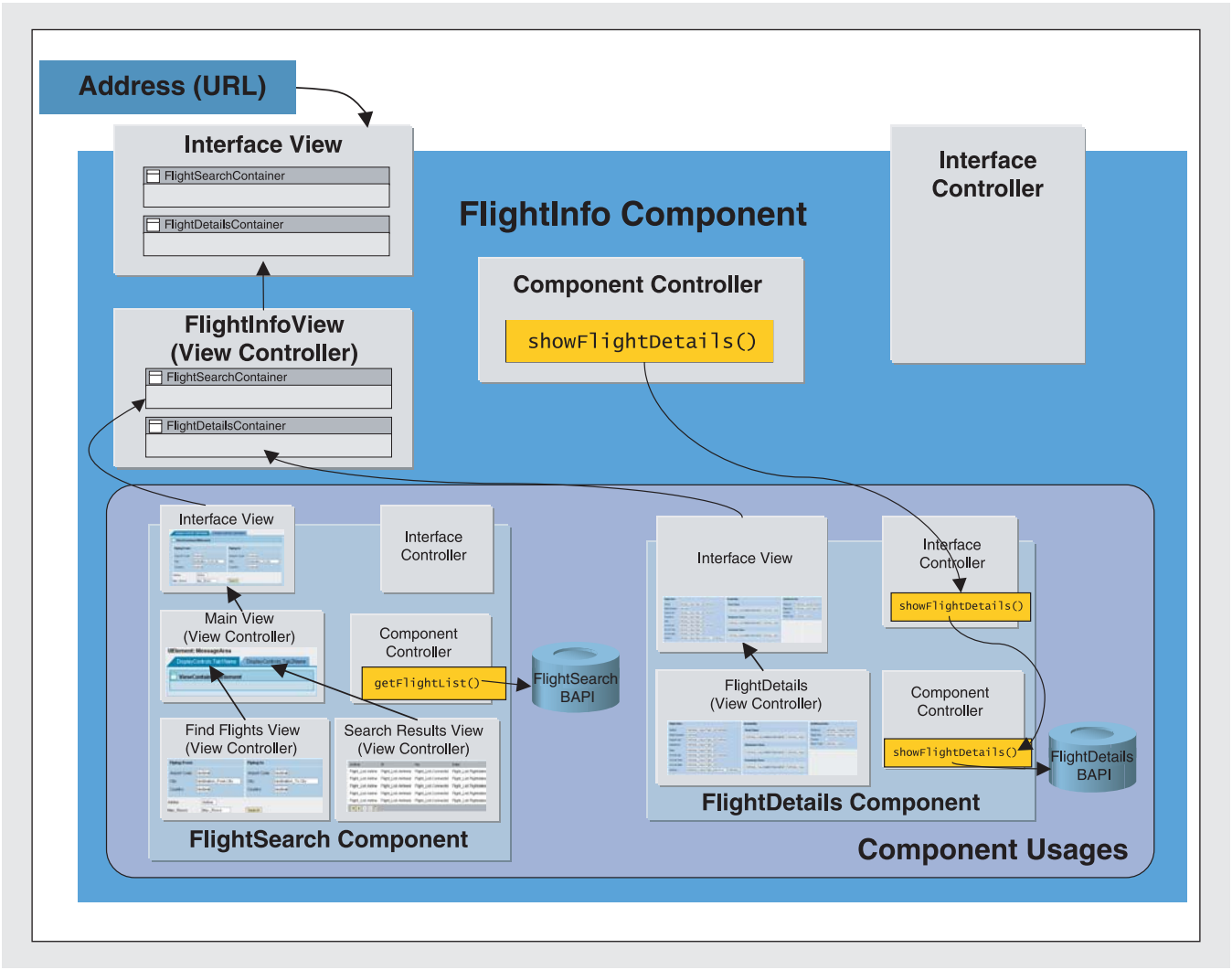

Figure 22 An example of the structure of a Web Dynpro application

front-end technology is used to display the business data.[28]

- The Web Dynpro application does not care what communication technology is required to access business functionality found in the back-end system.[29]
- Web Dynpro applications can be developed in either the Java or ABAP programming languages.

[28] Within the boundaries of SAP-supported front-end devices.

[29] Currently supported communication technologies include Remote Function Call (RFC) for ABAP function modules such as BAPIs, Web services (SOAP), and Enterprise JavaBeans (RMI).

Finally, the most important part — the fundamental principles of the Web Dynpro design philosophy:

- All Web Dynpro development is packaged into reusable units known as components. The component is both the unit of development and the unit of reuse.
- Try, as much as possible, to design each component so it represents an atomic unit of business processing.
- Always design Web Dynpro components with reuse in mind. Remember, coding is bad — if you have to write the same piece of code twice.

- At its heart, the Web Dynpro design philosophy separates those parts of the program that *generate* data from those parts of the program that *consume* data.

I trust that these concepts have been communicated with sufficient clarity and simplicity so you have an understanding of both the big picture of Web Dynpro and the architectural principles behind the design of Web Dynpro applications.

Chris Whealy started working with SAP software in 1993 making assembler modifications to the RF and RV modules of R/2. He then went on to work as a Basis consultant installing and upgrading R/3 systems, starting with R/3 version 2.0B. In May 1995, he joined SAP (UK) as a Basis consultant and ABAP programmer; however, when the Internet boom started in 1996, he turned his attention to Web-based interfaces to SAP systems. This led to work with the earliest versions of the Internet Transaction Server (ITS), and consequently, he taught the first course on this subject in January 1997. Since then, Web-based front ends for SAP functionality have been the main focus of Chris' attention. In January 2003, he started working with Web Dynpro and has worked closely with the development team in Walldorf, both learning the product and writing proof-of-concept applications. The knowledge gained while working with the developers became the foundation for the book "Inside Web Dynpro for Java" published by SAP PRESS in November 2004. Chris now works as the Web Dynpro expert for the SAP NetWeaver Regional Implementation Group (RIG) EMEA in Walldorf, Germany. You may reach him at chris.whealy@sap.com.

