Introducing the Enhancement Framework — a new way to enhance SAP programs without having to modify them

by Thomas Weiss and Michael Acker

Thomas Weiss
Product Manager,
NetWeaver Product Management Application Server,
SAP AG

Michael Acker
Development Architect,
NetWeaver Foundation ABAP Workbench,
SAP AG

(Full bios appear on page 128.)

A large set of customizing options enables SAP customers to adapt SAP programs to their needs. Customization alone, however, cannot always fulfill all customers' wishes for individual adjustments or enhancements to meet specific requirements. In such cases, programmers can enhance or change the standard SAP functionality by modifying the ABAP source code of the underlying development objects. This provision to modify standard capability provides a major advantage of SAP systems.

Up to now, there have been two methods available for meeting SAP software adaptation needs that cannot be addressed by customization alone:

- Modifications to SAP development objects either with or without the support of the Modification Assistant
- Enhancements to SAP development objects at predefined locations in the source code using customer exits, appends, includes, and as of SAP R/3 4.6, Business Add-Ins (BAdIs)

SAP NetWeaver 7.0 (formerly 2004s) offers the new Enhancement Framework that not only is intended to unify the modification and classic enhancement techniques, but offers you almost the same flexibility as modifications without the limitations of modifications. The new Enhancement Framework, which is integrated directly into the ABAP Workbench, enables you to change and enhance the SAP source code *without modifying it*. So you can have your cake and eat it, too.

The new Enhancement Framework significantly reduces the number of adjustments necessary after an upgrade because your enhancements survive the upgrade. With modifications, the changes are overwritten by the upgrade and must be reapplied. With the new enhancement approach, some adjustments still may be necessary if, for example, the upgrade changes the enhanced object in a way that is incompatible with the enhancements, but the reduced number of adjustment leads to a strong

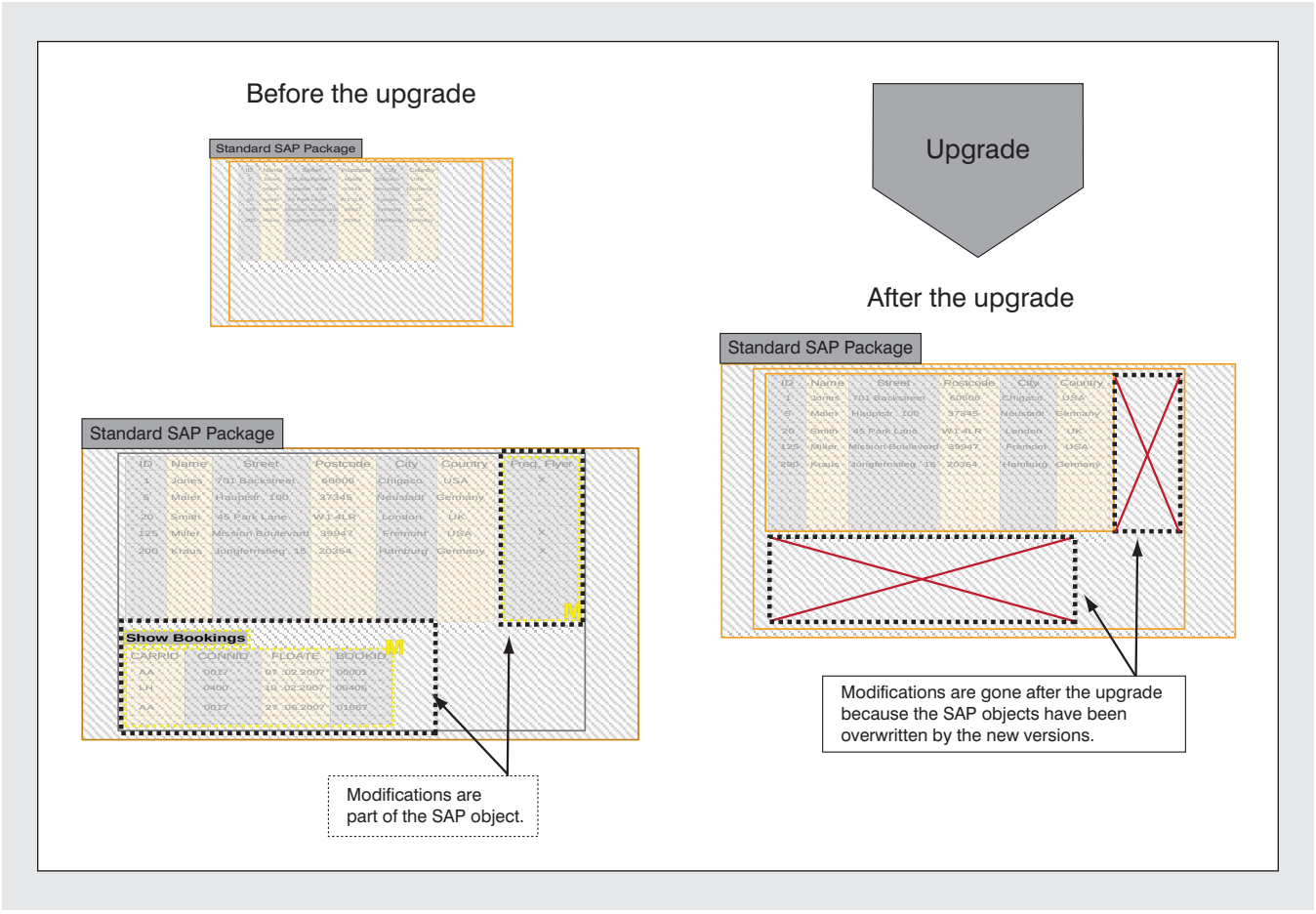


Figure 1 In an upgrade all modifications get lost

overall reduction in your total cost of ownership (TOC).

In this article we present an overview of the new Enhancement Framework. First, we review the limitations of modifications and the benefits of enhancements. With this background, you'll be better able to understand and appreciate the core idea of this new framework and how best to utilize it to enhance SAP compilation units or replace parts of these units. The Enhancement Framework enables you to organize both the enhancements and the "hooks" to which they are attached. This organization is required by the system, and we show you how to handle it and why it is not only useful, but also necessary in a real-world project. At the end of this article, we show you how to design a small example enhancement project. With this example, we will point out why and how even small semantic changes require quite a number of different enhancements. This motivates the careful planning of an enhancement project and underscores the necessity to organize the different enhancements carefully.

Limitations of the current modification concept

Even up to now the enhancement and modification support of SAP has been unique: To our knowledge, it is a unique selling point that SAP delivers the sources of their applications to the customer and, over and above, provides a controlled technology to modify

and enhance these sources and supports these procedures with particular tools. Although current SAP modifi-cation tools facilitate the handling of modifications with flexibility, you pay for this flexibility with some problems that are inherent in the very concept of a modification. Most of these problems center around upgrades and transporting modified units. Using **Figure 1**, let's take a deeper look at the limits of modifications to help gain an understanding of the advantages of enhancements: With enhancements, you can not only avoid most of the upgrade problems associated with modifications, but you can also profit from additional useful benefits that stem from the very structure of enhancements.

A *modification* means a change in SAP source code; that is, you add to, delete, or replace the source code delivered by SAP. Obviously, this leads to some drawbacks when you upgrade or patch your SAP system. In an upgrade, all SAP programs are replaced by new versions, whether they have changed or not. What does this mean for your modifications? Even if the SAP-supplied source code of the program is the same as it was before the upgrade, your modifications are effectively removed from the program after the upgrade.

Of course, SAP offers some support to help you reinsert your modifications after an upgrade. For instance, there is the modification adjustment transaction SPAU that supports many, but not all, types of objects (it does not support tables, transactions, or Web Dynpro ABAP, for example).[1] This transaction shows you, for example, a list of the parts of source code that have changed with the upgrade, and enables you to reinsert your custom code portions.

To which extent the Modification Assistant supports the automatic adjustment of modifications, and what the limits of this tool are, can be understood best through an example:

Imagine that you have modified the method `to_store` in the SAP program `store_admin`. What happens in an upgrade? First, the old version of the program `store_admin` is replaced by a new one. The manner in which you are able to reinsert your modification depends on whether the method `to_store` containing your modifications was changed:

- If the method `to_store` was changed in the upgrade, the SAP system cannot know, on its own, the position of your modification in the new version of this method. So you have to insert it again, semi-automatically. That is, you have to tell the Modification Assistant where to reinsert the modification.
- If the method `to_store` is the same after the upgrade, the situation is easier to handle. In this case, you can just click on a button in transaction SPAU and your modification is automatically inserted into the new version of the program.

While you can adjust many modifications automatically (those in modularization units[2] that have not changed) or semi-automatically (those in modularization units that have changed), as you will soon learn there are some situations where you will have to reinsert modifications manually. In many cases, transaction SPAU can serve as a reminder of which modifications you have to consider and a useful tool with which to reinsert them, but you still may have to go into the modified programs to locate the insertion point.

Let's now take a look at the main limitations of modifications:

- **In general, you have to adjust all modifications after an upgrade.** Otherwise your modifications would be gone after the upgrade. That means every modification causes work during an upgrade, even if the modified object is not changed in the next release.
- **You have to adjust some modifications manually.** The more complex the modifications get, the more difficult it is to manage or even keep the modifications. For example, if you transport modifications of the same unit into the same system from parallel developments in different

[1] For more on transaction SPAU and other modification tools, see the article "The Basics and Beyond: Manage Modifications Effectively with the SAP Modification Assistant, Modification Browser, and Object Adjustment Tools" (*SAP Professional Journal*, May/June 2003).

[2] Forms, methods, function modules.

systems, only the modification transported last survives. Suppose you have modification 1 of method `to_store` in system S01 and modification 2 of this method in system S02. If you transport both to system S03, only the modification that arrives last will be available in system S03. As soon as the last transport is integrated in S03, the other modification is gone. If you want to keep or merge both modifications, you have to manually reinsert the one that has vanished (and hopefully, you've kept good notes on all of your changes!).

Even more difficult is handling different modifications in different system layers. Whenever a modification is brought into a system that belongs to a higher layer, all modifications in the higher layer are lost. Let's say you have modification 1 of program `store_admin` in system S01 and modification 2 of this program in system S02, and that the transport route leads from S01 to S02. When the version of the program in S01 with modification 1 is transported to S02, it overwrites the version in S02 and modification 2 is gone. This is because modifications are part of the source code unit they modify, and a source code unit only exists once in a system. Again, to keep both modifications, you must manually reinsert the one that is lost.

- **You cannot arrange modifications independently.** Modifications can't be grouped, nested, or organized with a structure of their own. In transaction SPAU, all modifications are shown at the same level. It is possible to filter the displayed objects, but the remaining objects are all at the same level. This is, again, because modifications are not units of their own — *modifications are always part of the unit they modify*.
- **You cannot attach documentation to modifications.** Another limitation is that it's not possible to attach documentation to modifications. Of course, you can document your modifications directly in the source code or in another tool such as Microsoft Word, but both of these solutions are not as ideal as system documentation attached to the source code unit itself.
- **There is no easy way to track modification authors.** If you want to patch some part of a modification, it is always useful to know the original author. However, it is often difficult to keep track of the different authors of modifications. Transaction SPAU offers only limited multiple user support. Starting with SAP Web Application Server (AS) 6.10, all users who modify an object are logged, but it's not possible to ascertain from the log data who modified which part of the object.

In summary, the capability to modify SAP programs to meet your needs offers you significant flexibility, but it generates additional work during an upgrade, even with the support that transaction SPAU offers. It is not possible to merge or keep modifications from different developments of the same unit, nor is it possible to group and document modifications in the system. To address these potential drawbacks, SAP offers an alternative technology for changing system code: *enhancements*.

The main advantages of enhancements

In all the respects just mentioned, enhancements are more powerful than modifications, even given all the modification support that transaction SPAU offers. The reason is an important difference between an enhancement and a modification that stems from the conceptual difference between the two technologies:

- **Modifications are part of the unit they modify.** This is why modifications are first overwritten during an upgrade. As you have already learned, if you change objects that belong to the SAP namespace, these objects are replaced during an update.
- **Enhancements are objects in their own right.** If you, as a customer, write an enhancement, it belongs to *your* namespace, not to that of SAP. As a matter of fact, it belongs to a package of yours and is your transport object: Customer enhancements are customer objects. The fact that enhancements are objects in their own

right has a few important consequences. It is possible to:

- Transport enhancements separately from the compilation unit they enhance
- Keep or merge enhancements from developments in parallel or subsequent systems
- Group, or even nest, these objects, so that developers can easily view the available enhancement possibilities
- Attach documentation to these objects, so that additional information about particular enhancements is readily accessible without cluttering the source code
- Track who made which changes to what objects

Past SAP releases have offered enhancement technologies such as appends, includes, customer exits, and BAdIs. The Enhancement Framework is intended to integrate these technologies, improve upon them, and add some new ones to address recent developments such as Web Dynpro. SAP has developed this framework to fully take advantage of all the opportunities enhancements offer. For more overall information on the Enhancement Framework, see the sidebars below and on the following page.

In the upcoming sections, we'll take a closer look at the framework technologies, but first let's look at the underlying concept of the Enhancement Framework.

The Enhancement Framework concept: Enhancing objects without modifying them

At the core of the Enhancement Framework is the concept of an enhancement. The basic idea might at first sound a bit paradoxical: Making an enhancement is like modifying an object without a modification. While an enhancement behaves in many respects like a modification, it is no modification, and hence presents almost none of the drawbacks of modifications.

So how do enhancements without modifications work? At some positions of a development object you can enhance the object. These positions are called *enhancement options*. Such an option is like a hook for an enhancement. It is where you can attach the *enhancement* or, to use the correct full term, the *enhancement implementation element*. Doing this, you do not modify the compilation unit that you are

Enhancement Framework already tested in thousands of cases

The Enhancement Framework was originally introduced as a part of SAP ERP 6.0 (formerly mySAP ERP 2005) to integrate SAP Industry Solutions (IS) into the SAP ERP core.* It is this framework that enables the industry solution of SAP to move back to the core system. All the industry solutions have now returned to the ERP system in a large project in which some ten thousands of enhancement options and the respective implementations were created. Any particular portion of code of the ERP core that needed to be changed or enhanced by an industry solution was done by means of enhancements.

For all potential users of the Enhancement Framework this provides a great advantage: You use a new, but already proven, technology that has passed and is still passing a tremendously high number of tests under real-life conditions: Every running IS solution of the latest release has the Enhancement Framework inside and is empowered by it.

* See the *SAP Professional Journal* article "Introducing the switch and enhancement framework — consolidating industry solutions with the mySAP ERP core" (March/April 2006).

The Enhancement Framework is the technological backbone of the new Enhancement Package Strategy of SAP ERP.

With SAP NetWeaver 7.0, the Enhancement Framework was extended for use with the new Enhancement Package strategy of SAP ERP. This new strategy introduces a new dimension of choice in the world of business software:

- You can have all the advantages of staying on one release and still get new functionality. You can get the new functions not by upgrading, but as parts of Enhancement Packages. They provide new features on top of the recent release.
- For the new features, the import is separated from the activation. Directly after the import your system behavior does not change at all. Once imported the code of the new features is still not compiled before the are activated or switched on.
- On both levels, import and activation, you can choose: Import only what you want and activate only parts of the imported new functions that you need.

enhancing. The implementations you insert are processed at the appropriate position in the unit at runtime, but as transport objects, they are objects in their own right and can, for example, belong to another package. See **Figure 2**.

A comparison might help to understand how enhancements work. Compare the enhancement technology to a wall storage unit system. These storage systems typically allow you to insert a variety of elements at various positions without drilling holes in wall unit. At each position you can insert a shelf or other element; the available positions are where hooks have been provided by the manufacturer. Different positions may offer different kinds of hooks, and each hook type fits exactly one kind of element that can be inserted there. Once you have attached an element, a shelf for example, it seems as if the element is really part of the wall unit, but in fact it is only attached to the parts of the wall unit by the hooks or holders.

In a similar way, the enhancements look and behave as if they were part of the unit to which they are attached. They are processed at the point where they are inserted, but, as a matter of fact, they belong to a unit of their own. The different types of wall storage unit elements described in our comparison correspond to different types of enhancements. It depends on the type of the enhancement option what kind of enhancement you can insert there. There are, for example, options for adding a source code plug-ins, an additional method to a global class, or screen elements to a Web Dynpro component, to name only a few. This means that you can't enhance Repository objects anywhere that you like — you can insert an enhancement only where there is an enhancement option.

As a result of this separation between the enhancement definition and the implementation, enhancements are processed at the appropriate position in the source code unit, but they themselves are not part of the unit. This structure allows you to adapt an object without making a modification — the only change to the source code is the insertion of the "hooks" to which the actual enhancements are attached. In contrast to modifications, customer enhancements are in the customer namespace, which means that they are not overwritten by an upgrade.

So, at the center of the Enhancement Framework there is a simple structure along with some basic concepts:

- Enhancement options are hooks for enhancements. Since they define the possibility to enhance an ABAP Repository object, they reside on the

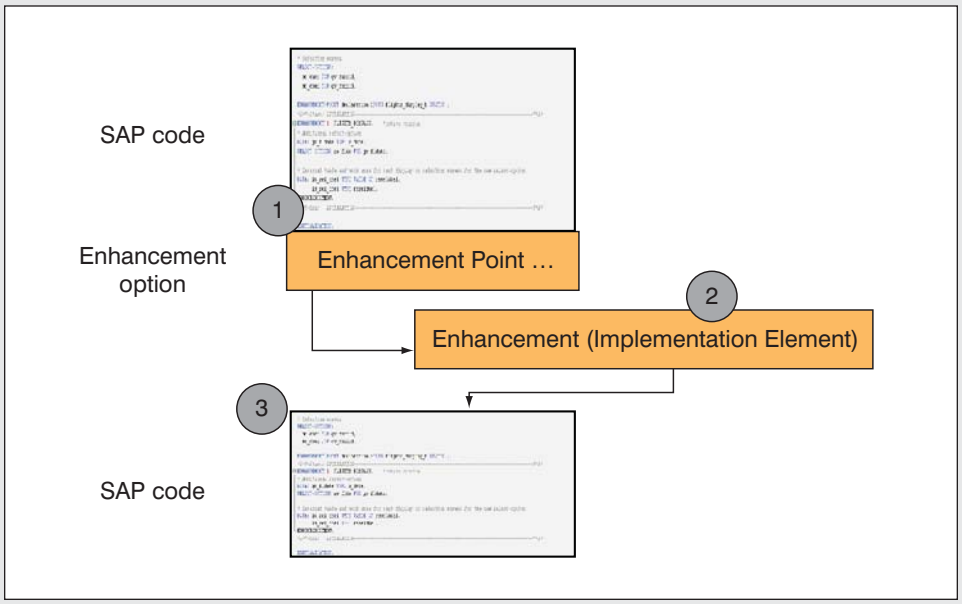


Figure 2 At runtime the flow control runs from the enhancement point (1) to the enhancement (2) and after the enhancement is processed returns to the code after the option (3)

definition side of the framework.

- Enhancement implementation elements are attached to those hooks. Since they actually implement the changes, they reside on the implementation side of the framework.

A framework that is intended to unify and standardize all the previous enhancement technologies is naturally quite complex, but as you can see, its basic structure is extremely simple. Let's now take a look at the technologies offered by the Enhancement Framework.

The Enhancement Framework technologies

Because you can only attach an enhancement where there is a hook, it is important to know the different kinds of enhancement options that are available.

In addition to the new kernel-based BAdI, there are many other types of enhancement options that are part of the new Enhancement Framework. You implement these enhancements using the Enhancement Framework tools, which are integrated into the ABAP Workbench — you simply switch to the *enhancement mode* directly within the ABAP Workbench tool that you are using, in the same way that you switch to the *change mode*.

Class enhancements

You can add methods to a class, or you can replace the code of a method entirely using an `overwrite` method. You can also add optional parameters to existing methods (added parameters must be optional so that the enhancement does not invalidate the code), or you can add `pre` and `post` methods to existing methods. A `pre` method is executed before the method to which it belongs and has a context of its own. This means that you have no access to the local variables

of the method. The same applies analogously to `post` methods. Using `overwrite` methods in conjunction with optional parameters gives developers a lot of freedom. The only part of a method that cannot be changed using the Enhancement Framework is the signature of a method specifying the mandatory parameters.

Function group enhancements

You can enhance the interface of a function module through the use of additional parameters. As with methods, you can add only optional parameters — but not mandatory parameters — because an enhancement must not invalidate the code it enhances.

Source code enhancements

There are two types of source code enhancement options — *enhancement points* and *enhancement sections*. Enhancement points and enhancement sections are positions defined in the source code where you can attach source code plug-ins that enhance the code. There is one notable difference between enhancement points and enhancement sections: Source code plug-ins that are inserted at an enhancement point are executed *in addition* to the existing source code, while source code plug-ins inserted at an enhancement section *replace* the code of the section.

Given these *enhancement sections*, not only you can add something to the original code, but you can also replace sections if they are so flagged by the developer of the unit. Viewed from the definition-side perspective, you can mark some lines of source code as an enhancement section, thereby offering the option to replace these lines in other systems without modification.

Web Dynpro enhancements

The Web Dynpro ABAP user interface (UI) technology[3] offers a variety of enhancement options. You can enhance a Web Dynpro view by adding UI elements to it — either new parts to existing elements (e.g., a new column to an existing table) or completely new elements (e.g., a new button). Many UI elements only make sense if they are "wired up": a new button should trigger some action, a new table should show some data, and a new input field should transport data to the business layer, for example. The Web Dynpro enhancements provided by the Enhancement Framework enable you to easily add such functionalities. You can:

- Add methods in the view or component controller and enhance existing ones using `pre` and `post` exits that are executed before or after the respective method is processed.

[3] Web Dynpro is based on the Model View Controller (MVC) design paradigm, where the application is separated into parts that generate data (the models), consume data (the views), and manage the communication between the generators and consumers (the controllers), all of which are encapsulated in a larger unit called a Web Dynpro component. For a detailed introduction to Web Dynpro and its concepts, see the article "Web Dynpro — what it is, what it does, why it exists, and how to get the best results from it" (*SAP Professional Journal*, January/February 2007).

Note!

In an SAP Web Dynpro ABAP view, you can overwrite all SAP-defined UI elements without modifying the underlying source code by deleting them in the enhancement mode. With this approach, the deleted elements still exist as transport objects, but are not compiled if the relevant enhancements are active, and you can populate the same space of the UI with elements of your choice. This way the Enhancement Framework enables you to realize almost every adaptation of a given SAP Web Dynpro ABAP UI without a modification.

- Add actions in the view controller and events in the component controller that trigger new methods if they subscribe to these new actions and events.
- Add plugs that enable you to create new navigation paths in the view controller.
- Add attributes in the view or component controller to hold data.
- Add nodes and attributes to existing nodes to hold the data for new UI elements.

The new BAdI

The new BAdI is now part of the Enhancement Framework. A BAdI is an object-oriented enhancement option, a hook for an object plug-in, and this way the most sophisticated enhancement type. When defining a BAdI, you determine its interface — that is, which methods the BAdI offers. BAdI implementations are classes that implement the BAdI interface. Typically, these implementations are created in another development layer; for example, objects owned by the customer that are in the customer namespace, while the BAdI definition and the call of the BAdI methods belong to the SAP namespace. Calling a BAdI method is, in many repsects, like a dynamic method call with a specified interface. Which methods are actually called is determined by the active BAdI implementation.

The Enhancement Framework introduces a new BAdI with several advantages over the classic BAdI introduced with SAP R/3 4.6 — although the basic functionality has not changed. Due to its integration into the kernel, the new BAdI has taken a big leap in performance and is directly supported by the ABAP language through two new commands `GET BADI` and `CALL BADI`, which make for an easier syntax when you create a BAdI instance and later call BAdI methods. There is also advanced filter support, which provides for more flexibility.

Note!

A typical use case of a BAdI is to provide room for a calculation that should be executed differently in each country. For example, an SAP application developer calls the BAdI method `calc_income_tax` to which he passes the income and wants to receive that total income tax, after he has defined a suitable BAdI. By taking advantage of BAdI technology, this developer doesn't need to know any details about the income taxes in the different country. He can rely on the fact that on another software level another developer with specialist knowledge will implement the BAdI method `calc_income_tax` in a suitable way when localizing the application.

It is the BAdI filter that manages the selection of the relevant implementation for each country. BAdI filters allow you to specify that only the implementations fulfilling the filter condition defined in the BAdI call are executed.

Note!

Keep in mind that the integration of the new BAdI into the Enhancement Framework also means that when you create and implement a BAdI, you have to conform to the structure required by the Enhancement Framework. Both a BAdI definition and a BAdI implementation have to be part of the relevant containers that will be explained later in this article.

Now that you have a solid understanding of the basic concept behind the Enhancement Framework

and the technologies that constitute the framework, let's take a closer look at the inner workings of enhancement options, which are the keys that unlock the functionality of the framework.

Enhancement options in detail

Since enhancement options serve such a crucial role in the framework, it's important to understand how they work. There are two types of enhancement options and two roles involved in their use. Let's begin by examining the two types of enhancement options.

Implicit and explicit enhancement options

Some things in life are free, and, of course, some are not. This is also true for enhancement options, which come in two flavors: *implicit* enhancement options, which are provided by the framework, and *explicit* enhancement options, which must be provided by the developer. Let's look at these two enhancement option types in some more detail:

- **Implicit enhancement options** exist in a unit without the developer of the unit having to lift a finger. There is no need to do anything explicitly — these are the ones you get for free. Implicit enhancement options are included with programs, global classes, function modules, and includes. The developer of a program or class need not make any preparations to make these compilation units enhanceable. This is done by the framework. Implicit enhancements comprise class enhancements, function group enhancements, Web Dynpro enhancements, and predefined enhancement points at the end of a report, a function module, an include, or a structure, and at the beginning and the end of a method.

- **Explicit enhancement options** require some work on the part of the developer, who must explicitly insert them in the compilation unit to make it enhanceable by somebody else. Only if the developer of a unit has preconceived the respective enhancement option can somebody do enhancements there later. Source code plug-ins and BAdIs are explicit enhancement options. You can, for example, insert an option for a source code enhancement with the commands `ENHANCEMENT-POINT` and `ENHANCEMENT-SECTION`, so that somebody else can insert a source code plug-in. BAdIs are a bit more complex. You must first create a BAdI in the BAdI Builder (integrated into transaction SE80), provide an instance of this BAdI in the source code, and can call a method of this BAdI.

As you just saw with explicit enhancement options, both BAdIs and enhancement points/sections must first be created by the developer so that somebody else can implement them: that is, there are two roles involved here, one for providing enhancement options and another for implementing them. Let's take a closer look at these roles.

Note!

Typically, the provider of an enhancement option is a SAP application developer who anticipates that the customer or provider of an industry solution might want to add or substitute some code at a particular point.

Another typical option provider is an independent software vendor (ISV) building some solution on top of the SAP NetWeaver AS ABAP. If this ISV wants to enable its customers to adapt its solution later to their own needs, it can do so by providing suitable enhancement options.


```
REPORT example_report
* some code
...
START-OF-SELECTION.
* Select Data
ENHANCEMENT-SECTION bselect SPOTS flights_display.
  SELECT carrid connid fldate price currency
    FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_carr
      AND connid IN so_conn.
END-ENHANCEMENT-SECTION.
...
```

Figure 3 An example enhancement section defined in the source code

Implementing and defining enhancement options: two different roles

Not only is defining and implementing an enhancement option technically different, there are two roles involved in enhancement options — somebody defines or provides the enhancement option (the *option provider*) and somebody uses the option to implement an enhancement (the *option implementer*):

- The **option provider** creates an enhancement option that provides the hook to which others can attach an enhancement. In general, the option provider is the developer of an object who anticipates that a user might want to add some code at a particular point to adapt the object to his or her own needs, and inserts an explicit enhancement option for this purpose. Perhaps a user of your program may want to select more columns than you normally do, so you add an option for a source code plug-in to the `SELECT` statement. Since it is not possible to insert an enhancement point within a statement, you must mark the whole `SELECT` statement as an enhancement section, allowing the implementation contained in the source code plug-in to replace this section of code. Where there is no hook, no enhancement can be attached. (In the case of predefined implicit enhancement options, the framework is the provider.)

  Figure 3 shows how this might look in the ABAP Editor. As you can see, it appears as a

Note!

You may have noticed the keyword `SPOTS` in **Figure 3**. At the beginning of the article, we mentioned that the Enhancement Framework requires its elements to be organized. For explicit enhancement options, this means that they must be assigned to a spot. The enhancement section `dbselect` shown in the figure belongs to the spot `flights_display`, which is a transport object of its own. We'll go into more detail about spots in an upcoming section.

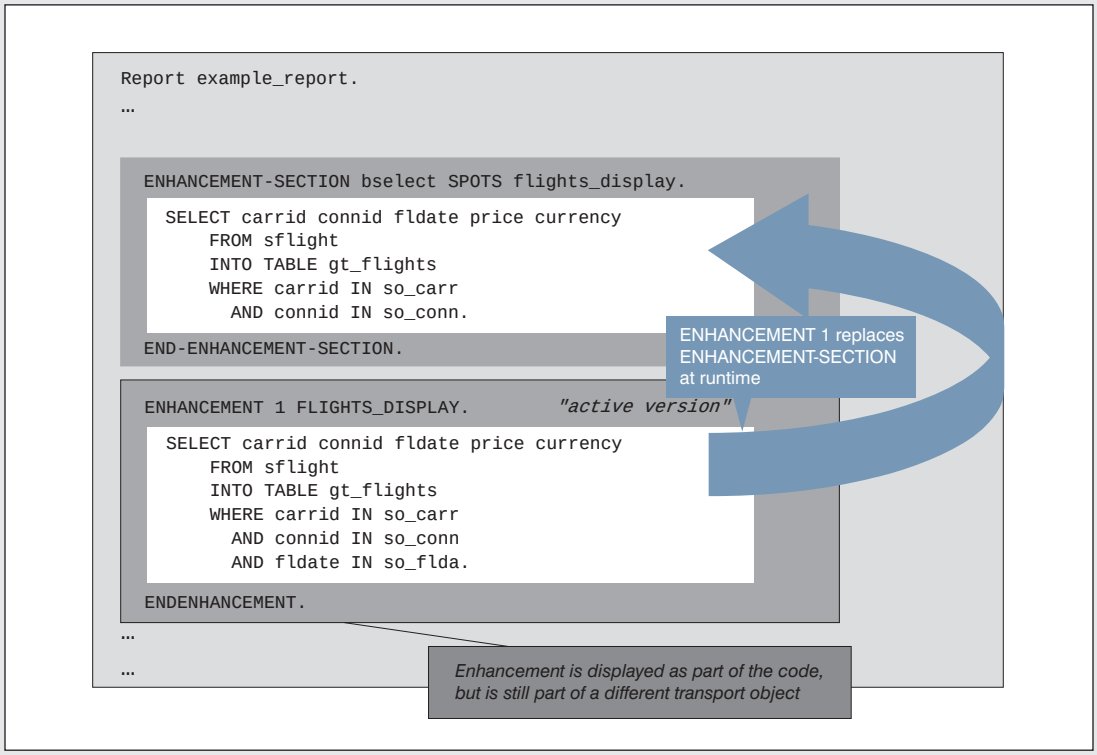


Figure 4 Implementation of the example enhancement section

normal portion of code. The two statements `ENHANCEMENT-SECTION` and `END-ENHANCEMENT-SECTION` function as brackets: The code enclosed by these statements is marked for possible substitution. As soon as the enhancement section is implemented by a source code plug-in, the new code contained in the source code plug-in is processed at runtime instead of the code between the statements.

- The **option implementer** is a developer who uses the enhancement option to implement an enhancement. An enhancement option all by itself, that is, without an implementation, does not do a lot when processed; in fact, it has no effect at all. It is a hook to which nothing is attached. So somebody needs to implement it if some work should be done here. The implementer doesn't need care about the details of an enhancement option, but does need to know how to use it. You might compare this to somebody who uses a service class; he or she only needs to know the public methods of the class or perhaps only the respective interface of interest, not what's actually inside of the class. A typical implementer is an SAP customer who wants to add some functionality, and above all, wants to do this without modifying the source code delivered by SAP.

Figure 4 shows what the implementation of our example enhancement section might look like. At runtime, the code within the source code plug-in shown in the lower half of the figure is processed instead of the code in the specified enhancement section. The implementer can implement such an enhancement section without *changing* the code

of the relevant report. There is a particular mode, the *enhancement mode*, to implement an existing enhancement option. So this needs not and cannot be done in the change mode.

These two roles (option provider and implementer) are not just conceptually different, but, in general, they are filled by different developers: Somebody provides a hook on *one* development layer so that somebody else can add or substitute some code there at *another* development layer. If a developer wanted to provide both the option and the implementation on one layer, he would not need the Enhancement Framework, but simply change the relevant development unit. In the case of an SAP customer working with a SAP object, this would, as a matter of fact, imply a direct modification of the SAP source code. Obviously, this is just what should be avoided with this new technology: What the whole Enhancement Framework is about are modification-free enhancements.

Note!

If there is no suitable explicit enhancement option around, an SAP customer should try to make do with the implicit enhancement options, which as already explained, are provided by the framework.

However, it is important that each role is aware of the other, even though they are separate. An option implementer needs some sound knowledge as to what the options they implement look like. The same applies vice versa to a developer defining an enhancement option. An option provider has to anticipate useful positions for an option, and can do this only if he or she knows how these options will be implemented.

Regardless of whether you provide an enhancement point or a BAdI, or implement an implicit or explicit enhancement option, you must follow the structure of the Enhancement Framework. We'll look at this next.

Spots and composites: Why and how to organize enhancement options and enhancements

No matter if you create an enhancement point or a BAdI as an enhancement option provider, or if you implement an existing enhancement option (be it implicit or explicit), at any rate, what you create must fit into the structure of the Enhancement Framework that enables you to organize the enhancement options and their counterparts on the implementation side. This means that the explicit enhancement options and all enhancement implementation elements (enhancements) have to be part of particular containers, which also serve as transport objects. In this section, we will explain what these containers are and how they are structured. At the end of it, you should know all you need about organizing enhancements options and enhancements.

At first, working with these containers might appear a bit cumbersome. For example, if you want to create and implement a single enhancement option, you might wonder why a container is necessary at all; it might look as if you could do with just the enhancement option and the respective enhancement implementation element. In fact, one enhancement will hardly ever suffice in a real-life project. Developing enhancements is different from writing a program from scratch. Its very nature leads to a larger number of enhancements:

- Many small enhancements create less work in an upgrade than a large one.
- One enhancement necessitates other enhancements.

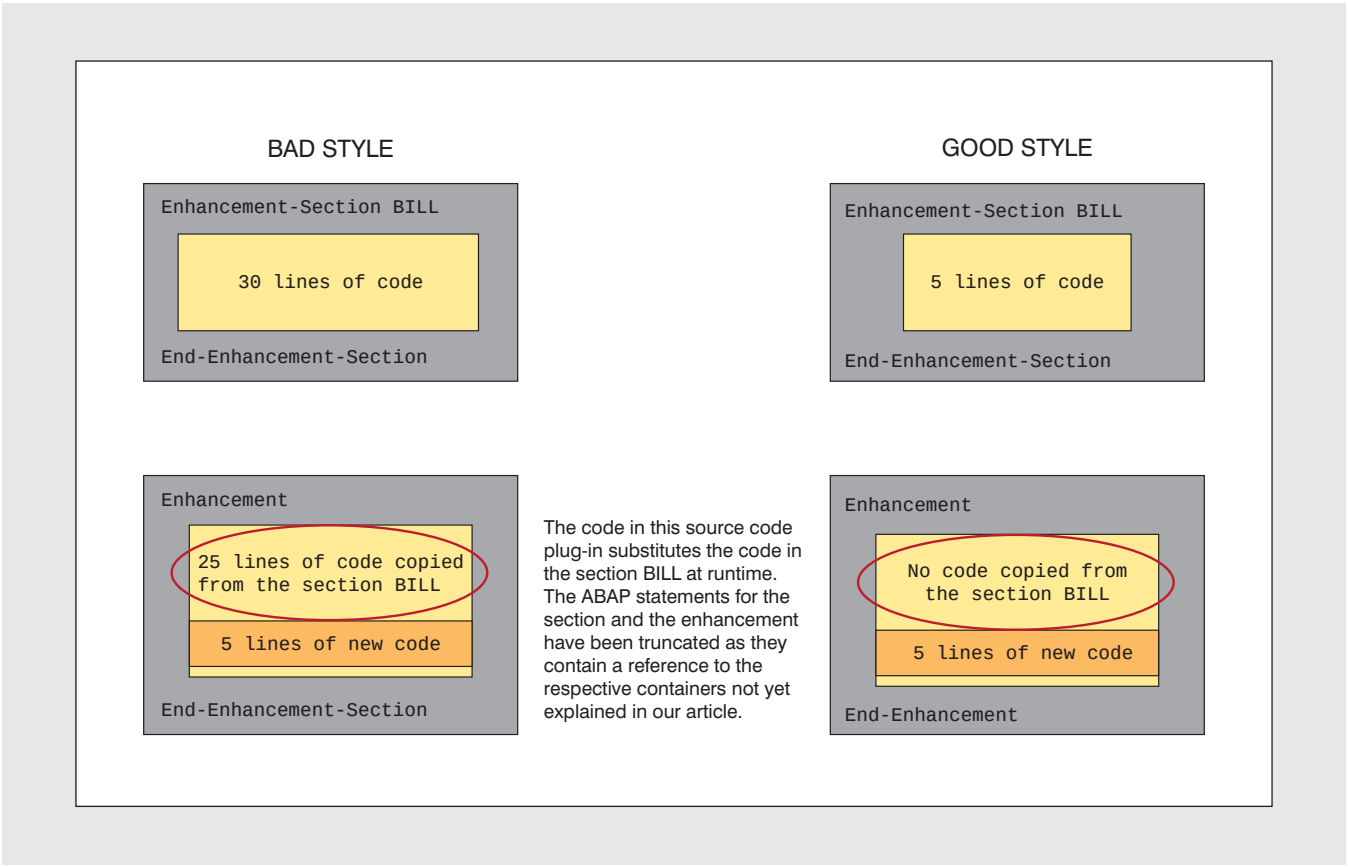


Figure 5 Use small enhancements rather than large ones

So let's look in some details at why you will end up with a large number of enhancements.

Several small enhancements are better than one large enhancement

If you keep your adaptations small instead of, for example, replacing an entire program, the chance increases that they will create less work in an upgrade. Remember from earlier in the article that one of the promises of enhancements is that they can survive an upgrade without requiring a lot of additional work. Even the comfort offered by the Enhancement Framework in an upgrade has a limit, however. At a minimum, the enhancement option — the "hook" for the enhancement — must survive the upgrade. And so must any prerequisites on which your enhancements depend. If you use a particular data field in an enhancement, you must not only rely on its declaration, but also be aware that semantically your enhancement may depend on some calculations being executed before the enhancement is processed. If the declaration vanishes in an upgrade, you get a syntax error. If the calculation your enhancement depends on gets lost in the update, your enhancement no longer makes sense. Therefore, even enhancements in the Enhancement Framework are not totally immune to changes caused by an upgrade.

The smaller a portion of code replaced by an enhancement is, the smaller is the possibility that the original code will be changed or deleted by SAP in an upgrade. So a number of small enhancements are less susceptible to changes of the enhanced object than one very large enhancement — there are less chances for dependencies that could cause problems after an upgrade.

Moreover, large enhancements often require you to copy code. And this should be avoided, because copied code generally causes more maintenance effort. So, for example, it is better to replace only a few lines of code in an SAP method implementation than to replace the entire implementation, as shown in **Figure 5**. Substituting a long section of code with a source code plug-in often requires the implementer to copy code from the original development object into the source code plug-in. If SAP changes the parts of the copied code in an update, the enhancement version has to be adjusted correspondingly, creating additional maintenance work. Keeping your enhancement small will save this work for the implementer.

While using small enhancements avoids the extra work and potential problems that a single large enhancement might create in an upgrade, a side-effect is that you can end up with quite a few enhancements to manage. But there is also another reason why, in general, you will end up with more enhancements than you might have expected.

Why one enhancement in general necessitates other enhancements

You enhance a SAP development object to achieve some aim on the semantic level. Usually this aim cannot be reached by one enhancement only. For example, suppose you have enhanced a `SELECT` statement by an additional field. First, you need to enhance the structure that holds the result set. And most probably the new field should do some work. If your software is properly divided into components, you have to enhance the interfaces of the relevant methods that work with the new data. You probably also have to add some additional calculations because the additional data needs to be validated and computed. And after all you want to present some additional value in the user interface. That means you have to add an additional field to your user interface and to transport the respective value there. In this way, enhancements are sociable beings — they hardly ever come alone. Making only a small semantic change by enhancing a piece of software, you will end up with quite a number of enhancements spread over different components of your software.

So the very nature of enhancement programming leads to a larger number of enhancements. And a larger number of objects need to be organized. In addition, there are the well-know reasons that require some structuring of every software project: Enhancement options may be created and implemented by a team, or even multiple teams, and be part of different projects. And you need to pigeonhole the enhancements in a way that mirrors the team and project structure.

So there are plenty of reasons why you will have many enhancements and why these many objects need some organizing. This is why the Enhancement Framework not only provides containers but also enforces a container structure on the different objects. Containers enable you to organize your enhancements and explicit enhancement options according to type, project, content-related matters, and other sorting criteria, and these containers also serve as transport objects.

Now that you know why enhancement containers are necessary, let's take a closer look at how they work.

The structure of the Enhancement Framework containers: Enhancement spots and enhancement implementations

Option providers and implementers must assign all explicit enhancement options (i.e., BAdIs and source code plug-ins) and all enhancement implementation elements, respectively, to containers.[4] Explicit enhancement options must be assigned to (simple) enhancement spots, enhancement implementation elements must be assigned to (simple) enhancement implementations, each of which can be grouped into composite enhancement spots or composite

[4] Implicit enhancement options, on the other hand, need not and cannot be grouped in containers.

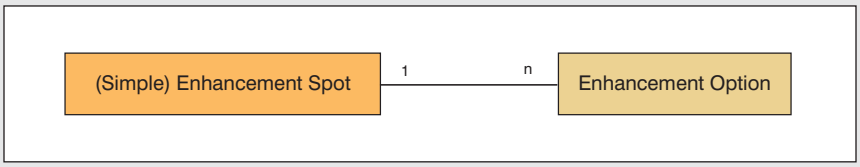


Figure 6 The relationship between a (simple) enhancement spot and an enhancement option

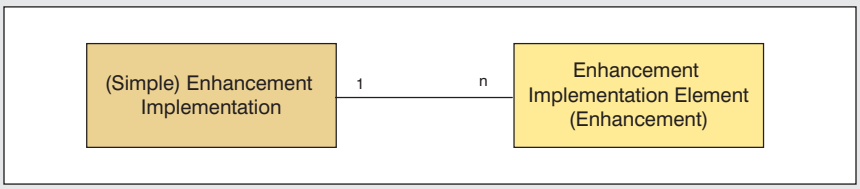


Figure 7 The relationship between a (simple) enhancement implementation and an enhancement implementation element

enhancement implementations, respectively.[5] These containers not only serve to manage your enhancements and enhancement options and keep them organized, but also serve as transport objects.

(Simple) enhancement spots and (simple) implementation elements

Explicit enhancement options and all types of enhancement implementation elements must belong to a container in the same way that a method must have a class. You can add a new method to an already existing class or create a new class for this method; either way, it must belong to a class. The same holds true for enhancement options and enhancements implementation elements. They cannot exist on their own. They need to belong to a (simple) enhancement spot or a (simple) enhancement implementation, respectively:

- **(Simple) enhancement spots:** You must create an explicit enhancement option within a (simple) enhancement spot. A (simple) enhancement spot can contain many enhancement options, while an enhancement option is uniquely assigned to one (simple) enhancement spot, as shown in **Figure 6**. A (simple) enhancement spot can only contain enhancement options of the same type; that is, a spot that contains enhancement options for source code plug-ins cannot also contain BAdIs.
- **(Simple) enhancement implementations:** You must create an enhancement implementation element within a (simple) enhancement implementation. A (simple) enhancement implementation can contain many enhancement implementation elements, while an enhancement implementation element is uniquely assigned to one (simple) enhancement implementation, as shown in **Figure 7**. As with (simple) enhancement spots, a (simple) enhancement implementation can only contain enhancement implementation elements of the same type.

There is also another limitation: A (simple) enhancement implementation containing source code plug-ins can contain only source code plug-ins

[5] The "(simple)" prefix signifies that it is an individual enhancement spot or enhancement implementation as opposed to a composite, which is a grouping of semantically related enhancements spots or implementations. In unambiguous contexts you can omit this prefix.

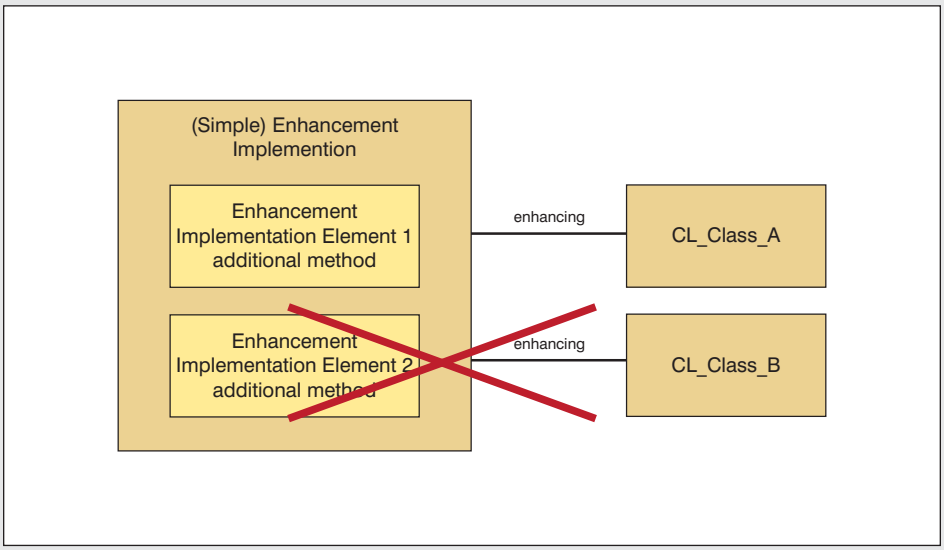


Figure 8 Class enhancements in one (simple) enhancement implementation must enhance the same class

enhancing one report or only additional methods and additional attributes enhancing one class (see **Figure 8**). Class enhancements in one enhancement implementation must belong to one class, function group enhancements to one function group, and Web Dynpro enhancements to one Web Dynpro component.

Composite enhancement spots and composite enhancement implementations

Nevertheless, there is the need to put all enhancements of a particular project in a single container: One of the main reasons for enhancement spots and enhancement implementations is to enable you to put entities semantically belonging together in one basket. But, as we just described, (simple) enhancement spots and implementations can only contain options or elements of the same type. So what do you do when you need to put all of the enhancements of a particular project, which are bound to be of different types, in a single container? There is another type of container for this purpose — composites:

- **Composite enhancement spots:** Composite enhancement spots group (simple) enhancement spots that semantically belong together — for example, enhancement spots belonging to the same semantic area — and other composite enhancement spots. The grouped (simple) enhancements spots can contain enhancement options of the same or different types. Composite enhancement spots can only contain (simple) enhancement spots directly, not enhancement options. The enhancement options belong to a composite enhancement spot through the enhancement spot that contains the options. Composite enhancement spots can also be nested, meaning that a composite enhancement spot can contain other composite enhancement spots.
- **Composite enhancement implementations:** Similar to composite enhancement spots, composite enhancement implementations group related (simple) enhancement implementations, which can contain enhancement implementation elements of the same or different types. As with

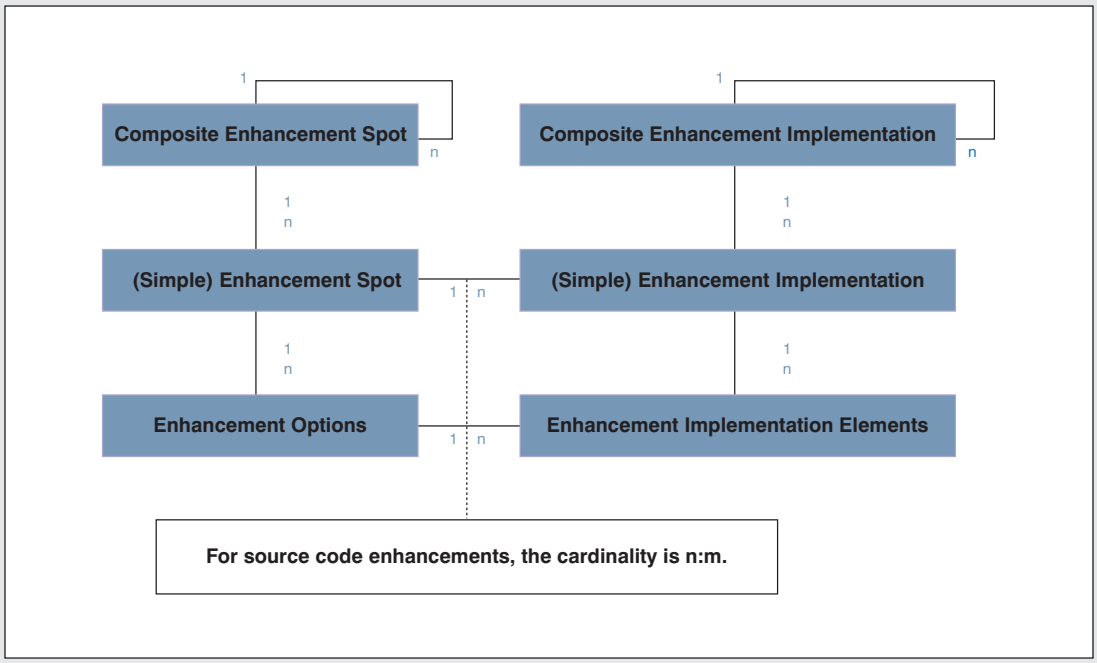
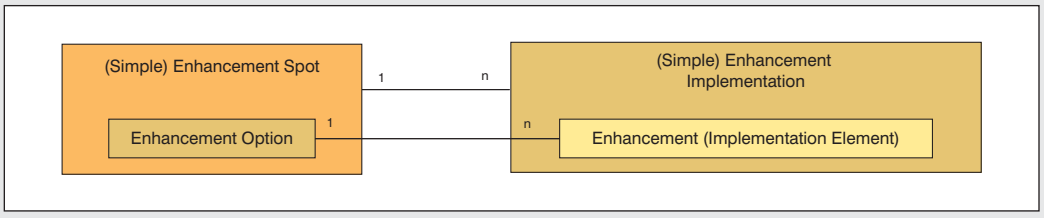


Figure 9 Relationships between enhancement spots, options, implementations, and implementation elements

The cardinalities in detail

As to the cardinalities concerning the relations between the definition side and implementation side, an enhancement spot for BAdIs can be assigned to many enhancement implementations, but an enhancement implementation for BAdI implementations uniquely belongs to an enhancement spot (cardinality 1:n). Spots and implementations for source code enhancements are assigned in a different cardinality (m:n).



composite enhancement spots, composite enhancement implementations can only contain (simple) enhancement implementations directly, not enhancement implementation elements, and they can be nested.

The new container types in brief

So there are four new transport objects in the new Enhancement Framework:

- (Simple) enhancement spots can contain explicit enhancement options of the same type (i.e., BAdIs).
- (Simple) enhancement implementations can contain enhancement implementation elements that are of the same type and implement enhancement options in the same development object.
- Composite enhancement spots can contain (simple) enhancement spots of the same or different type and other composite enhancement spots.
- Composite enhancement implementations can contain (simple) enhancement implementations of the same or different type and other composite enhancement spots.

Implementing an enhancement option: The relations between the implementation and the definition side

So far we have only described the relations on the definition side and those on the implementation side in some detail. But obviously, this does not suffice, because implementing an enhancement option means to assign it to an element on the implementation side. Doing this in the context of the structure of the Enhancement Framework is a bit tricky: You can only assign an enhancement (implementation element) to an enhancement option if the respective containers are also suitably assigned. So you have to have an assignment of the (simple) enhancement implementation to a (simple) enhancement spot, and it is only then that the elements in these containers, the enhancement option and the enhancement (implementation element), can be suitably assigned. For more information on cardinality assignments, see the sidebar on the previous page.

How it all fits together

Let's now look at all these objects of the Enhancement Framework and the way they are assigned to each other, **Figure 9** summarizes the relationships between all of these objects.

Because this structure as a whole is pretty complex, let's review the explanations of the main details:

One enhancement spot can contain many enhancement options, while an enhancement option is uniquely assigned to a spot. The same applies in an analogous manner to the relation between simple enhancement implementations and the enhancement implementation elements.

- The simple containers on both the definition and implementation side can only contain elements of the same type: A spot that contains BAdIs cannot contain enhancement points/sections and vice versa.
- The composite containers are not affected by this restriction: A composite enhancement spot can contain simple enhancement spots that in turn contain enhancement options of different types. One composite enhancement spot can contain many (simple) enhancement spots, while a (simple) enhancement spot uniquely belongs to a composite enhancement spot. Composite spots can be nested. But, of course, a composite container cannot contain definitions *and* implementations. There are composite enhancement spots and composite enhancement implementations. And this makes perfect sense: All the entities on the left hand side (the definition side) are created by the enhancement option providers, while those on the right hand side (the implementation side) are used by the implementers.

To give you a still better understanding of it, consider an example in which the relation between elements, containers, and high-level containers is

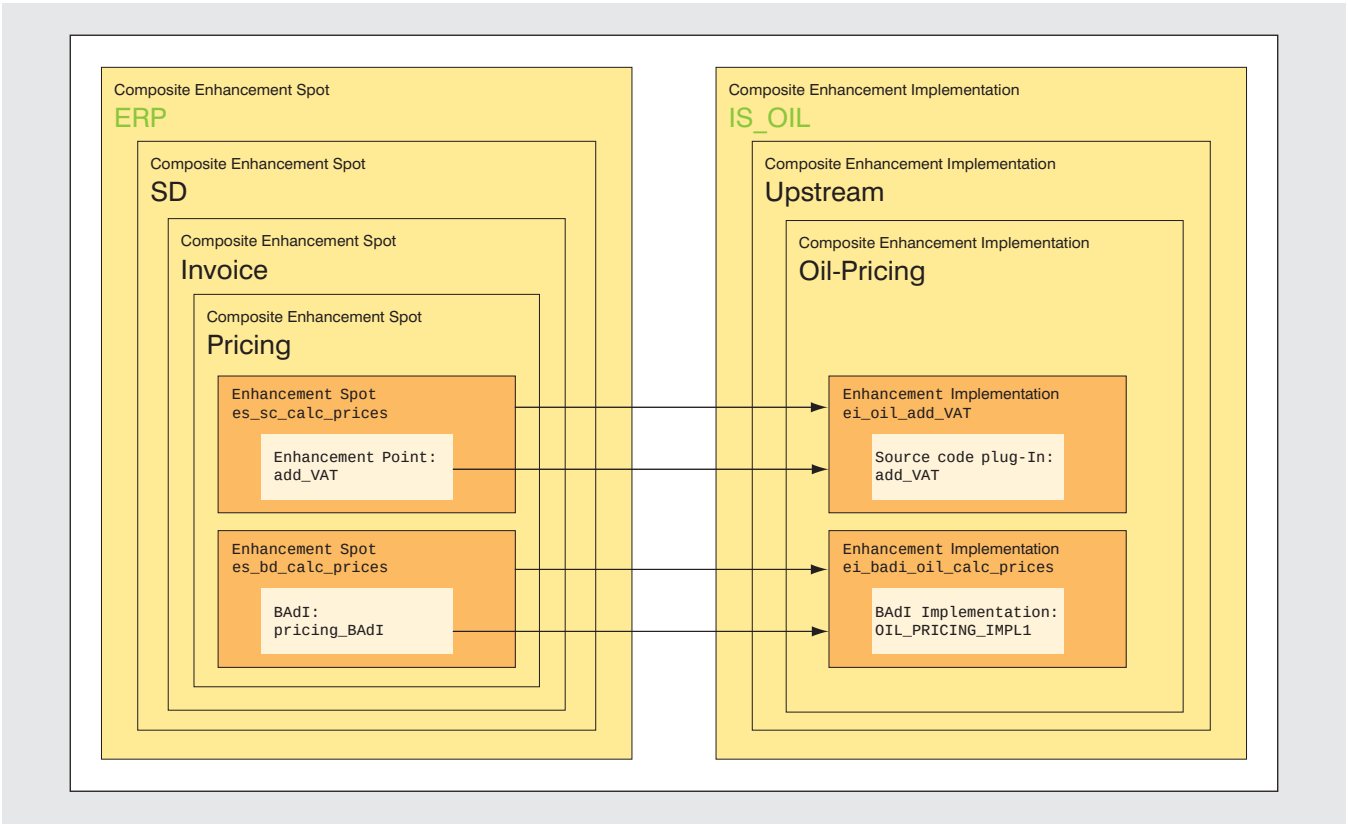


Figure 10 Relationships between enhancement spots, options, implementations, and implementation elements in an example

shown by the special inclusion. See **Figure 10** (this is only an example and does not show the names of real-world objects).

To sum it up, when you create an explicit enhancement option or any enhancement implementation element, it is mandatory that they belong to a container that is an (simple) enhancement spot or an (simple) enhancement implementation, respectively. This assignment is enforced by the respective tools. While it is not necessary to have composites for real-world projects, it is good practice to use them to keep your enhancements intuitively organized.

Now that you have a solid understanding of the concepts behind the new Enhancement Framework, let's make these abstract relationships more tangible by looking at an example.

The design of a typical enhancement project

In this section, we present a typical enhancement project using a small demo application. Walking through this demo application we will look at the number and type of enhancements that can arise, why you will probably end up with more, not with less enhancements than you might have supposed initially, how these enhancements interact, and how they should be organized. (We don't explain how to create and implement these enhancements in this article.)

An enhancement project needs a careful design, just like a development project. You might feel tempted to insert single enhancements here and there, just because it is so easy with the new Enhancement Framework. Adding to the code (by implementing an

implicit enhancement point or adding a `pre` or `post` exit to a method of a global class, for example) or changing the code (by using the `overwrite` exits for methods of global classes, for example) of SAP development objects can now be done with much less effort than with modifications. But without careful planning, even a single enhancement could thwart the internal logic of an application.

The risk is even greater with a large number of enhancements, which almost always ends up being the case once you start to enhance an application — remember, enhancements hardly ever come alone, and chances are you will end up with many more enhancements than you initially expected. Plus, with increasingly more unstructured enhancements, you might easily lose sight of what and where you have enhanced something. So, a carefully considered design and enhancement organization is critical to the success of your project.

Our simple example enhancement project is based on a small application that uses the well-known flight data model based on the SAP IDES training system. The application consists of two Web Dynpro ABAP components — one that displays a table of flight bookings, and another that enables the user to input a new booking. Both components follow the MVC programming model that separates the business logic from the presentation logic. While this design might seem a bit cumbersome in the context of such a simple example, we use it here to mirror the design of a real-world application, so that you can apply what you learn to your own applications.

Let's start by taking a look at the first component, which displays the bookings of a flight, and how we plan to enhance it.

The flight bookings display component

Figure 11 (on the next page) shows the flight bookings display component. Starting at the UI, the flight booking table `TABLE_BOOKING` is displayed as a UI element in a Web Dynpro view. `TABLE_BOOKING` is bound with the cardinality 0:n against the node `N_BOOKINGS`, which holds the data of the table. The data of the node is supplied by method `wdoninit`, which in turn calls method `provide_data` to retrieve the requested data from the back-end database table `BOOKINGS`. We use two different methods to retrieve the data to separate the presentation from the business logic. Note that the different layers all use the same ABAP Dictionary data type (`BOOKINGS`) for the parameters of data objects. This will turn out to be particularly useful when we enhance the component.

Figure 12 (on page 121) shows what the component looks like when we add a new column to the table. Let's look at the enhancements we need to make this work. Starting at the UI, we first add a column to `TABLE_BOOKING` using a Web Dynpro enhancement. Because this column needs some data, we enhance the respective node (`N_BOOKINGS`) by adding an attribute, also using a Web Dynpro enhancement. The table `BOOKINGS`, which is the basic data provider, gets an additional column/component (`INSEAT_SCREEN`) through a classic table append.

Because of this table append, the interface of the supply method `wdoninit` does not need an enhancement — it gets the new component (`INSEAT_SCREEN`) without anybody having to lift a finger. The same applies to the method `provide_data`. This is because the relevant parameters are typed with the ABAP Dictionary type `BOOKINGS`, and once an append is added to this type, every user of the type gets the new component automatically. This way, we get a broader interface for both methods for free.

We also need to enhance the `SELECT` state-ment of the method `provide_data` to select the data in the new column. Remember that you cannot place an enhancement point within a statement, so instead we replace the original `SELECT` statement with a new one that selects the data originally required for the table plus the additional component `INSEAT_SCREEN` by inserting a source code plug-in at the defined enhancement section.

Looking at this example, you can see how simply adding a new column to a table in the UI necessitates a bundle of other enhancements. It is

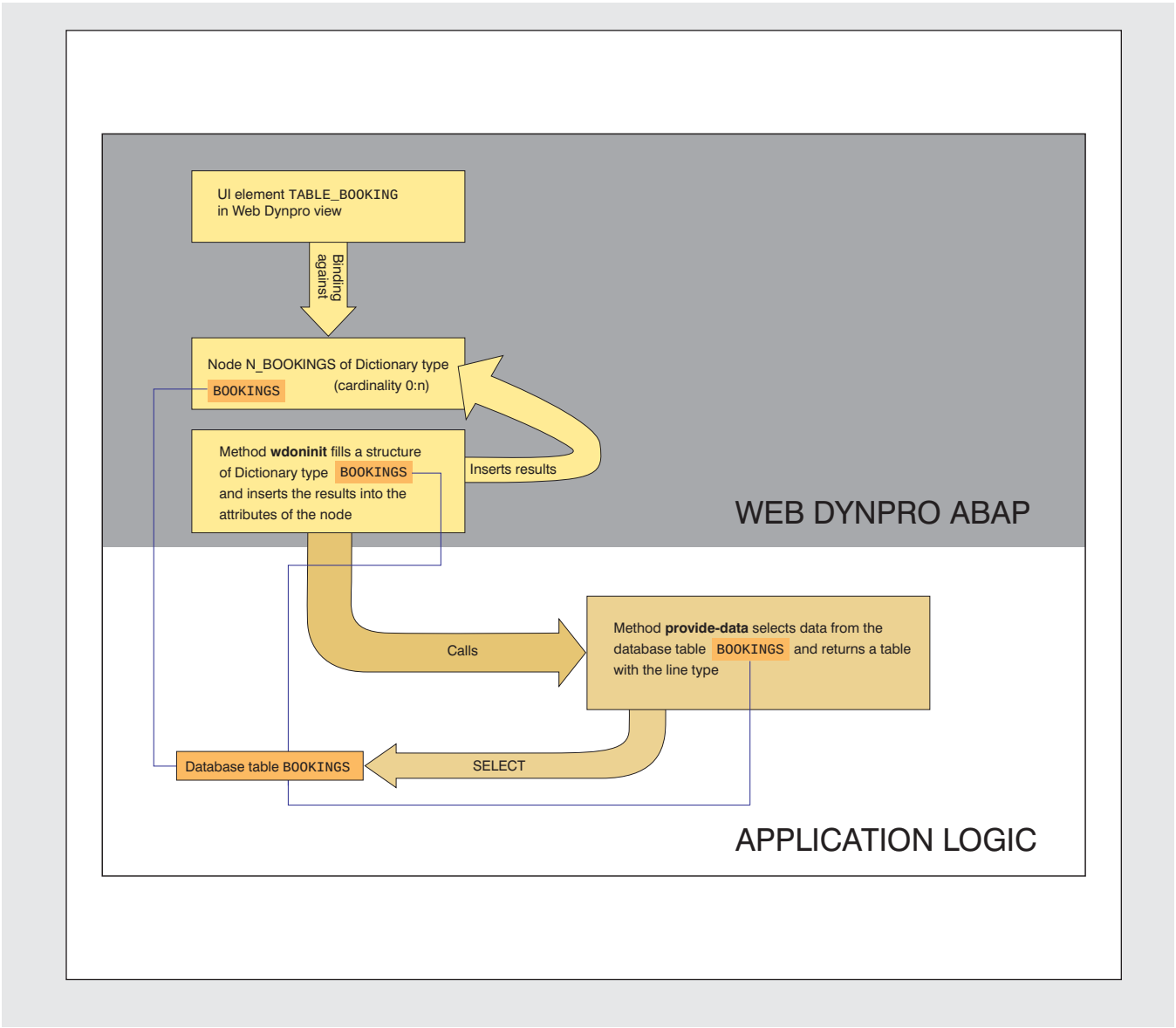


Figure 11 The flight bookings display component

Note!

If the developer of the original program had not marked the `SELECT` statement as an enhancement section, we would have substituted the entire implementation with an `overwrite` exit, which is an implicit enhancement, so it is always available. But this should only be our second choice — replacing an entire method just to adapt a `SELECT` statement would mean copying a lot of code from the original method to the `overwrite` exit, which causes more work in an upgrade. Remember that if any part of the code you have copied changes after an upgrade, you have to adapt your enhancement.

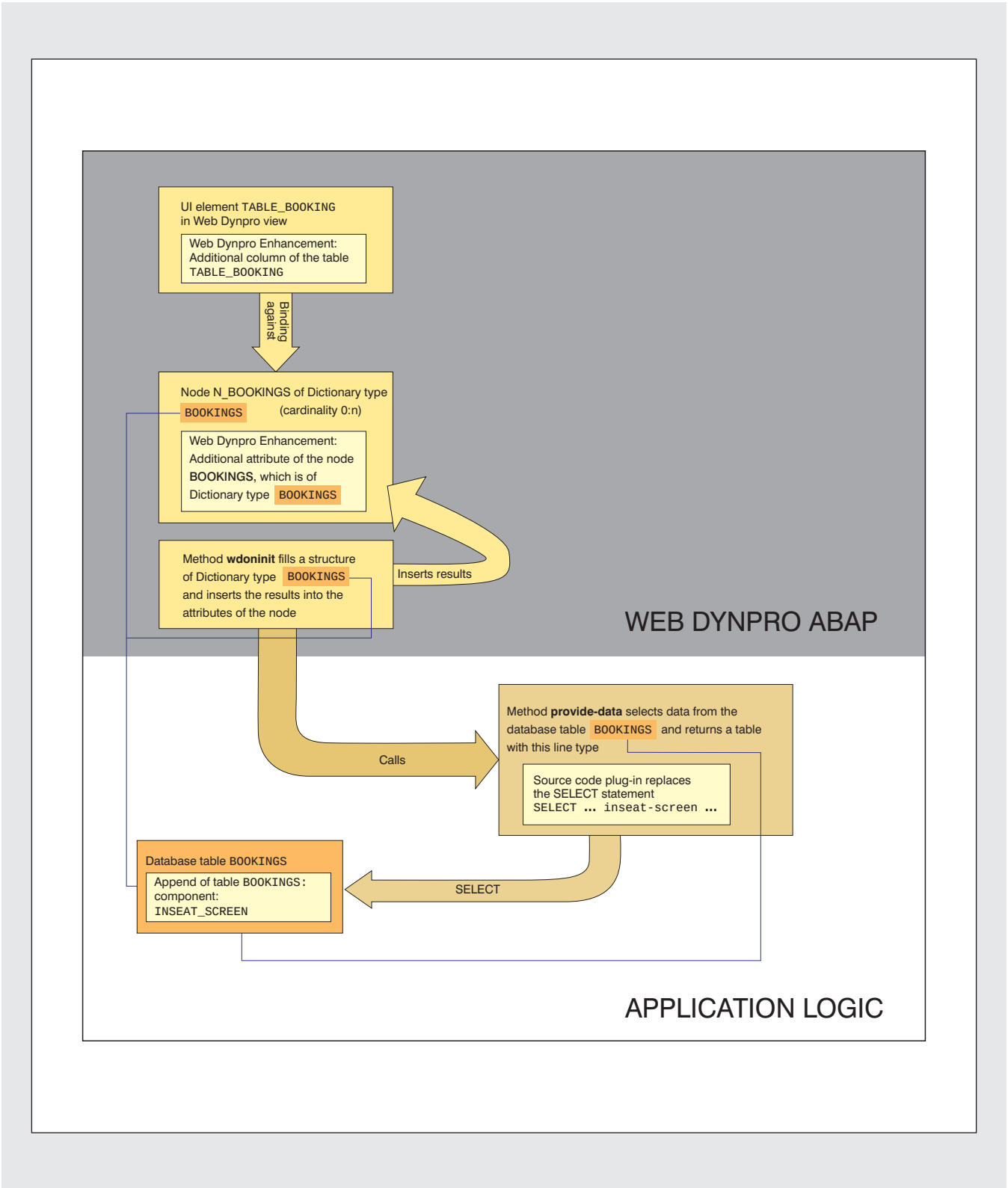


Figure 12 Adding a new column to TABLE_BOOKING

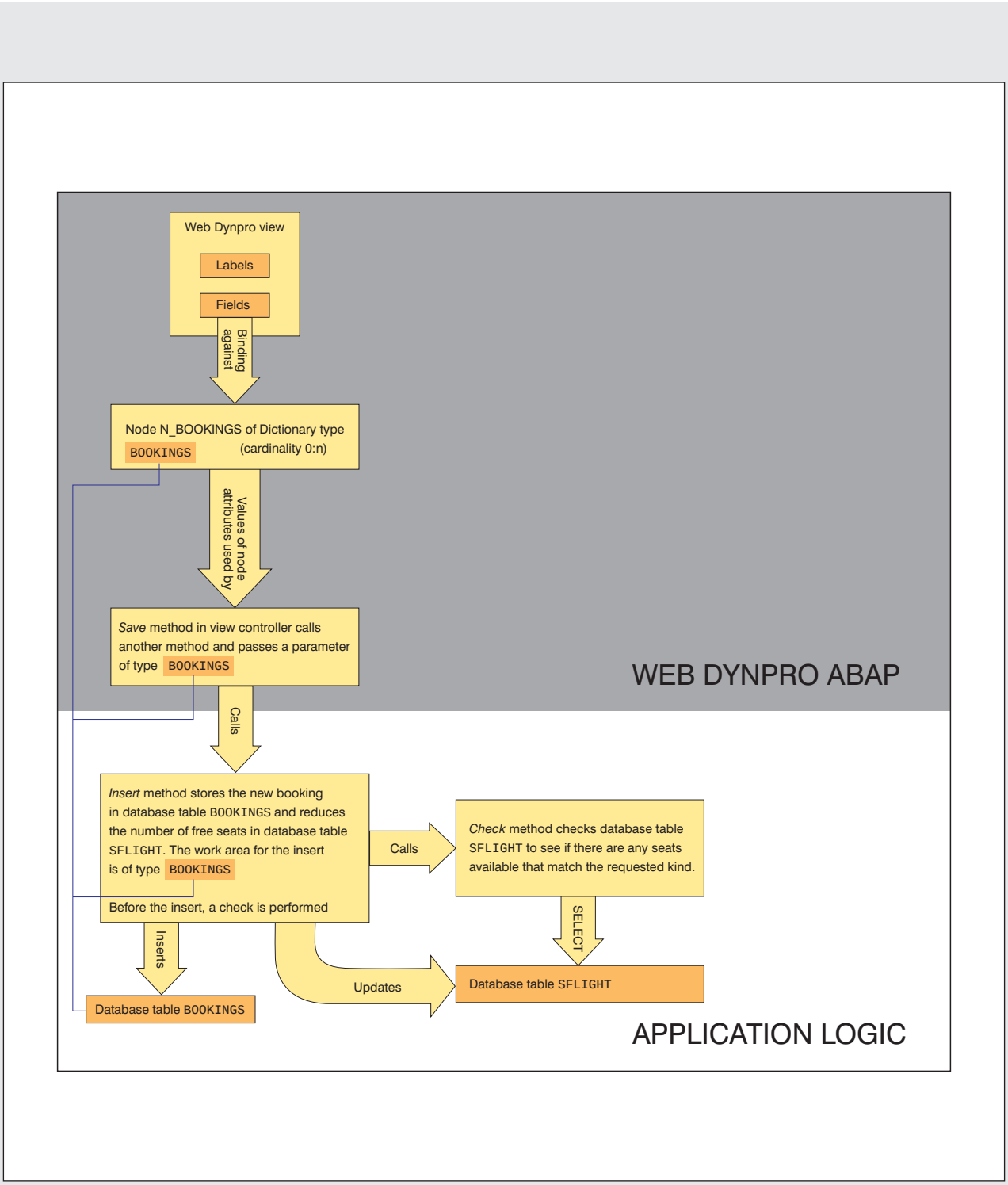


Figure 13 The user input component

only because the interfaces are typed as an ABAP Dictionary type that we do not need even more enhancements.

The user input component

The second component of the example application enables the user to input a new booking. **Figure 13** shows the key elements of this component (we show only some important elements in our diagram). There is a label and a field, respectively, for each component of the structure `BOOKINGS` to be input in the UI. These fields are bound with the cardinality 0:1 against the node `N_BOOKINGS` of the appropriate type `BOOKINGS`. Again, there are different successive methods that transport the value to the database: The method in the view controller calls an *insert* method: As in the flight bookings display component, there is the division between presentation and business logic. Before the new line is inserted into the database table, the program checks the table `SFLIGHT` to see if there are still seats of the requested kind available. If the check is successful, the table `BOOKINGS` gets a new entry and the table `SFLIGHT` is updated.

Now we enhance the component by adding the new field `INSEAT_SCREEN` plus a label to the Web Dynpro view and everything else we need to enable this new input field to do its job, as shown in **Figure 14** on the next page. We bind the new field — again with the cardinality 0:1 — against the node `N_BOOKINGS`, which is also enhanced using a Web Dynpro enhancement. The two methods that transport the data back to the database do not need an enhancement of their interface; as we saw with the first component, the interface of type `BOOKINGS` is enhanced automatically if this database table is enhanced in the ABAP Dictionary. The database table `SFLIGHT` needs four new columns so that we can keep track of the number of total seats and occupied seats with and without inseat screen. We replace the *check* method by implementing an `overwrite` exit because we now need a different check, since there are new fields in the database table `SFLIGHT`.

Again we see how database appends and enhancements interact perfectly and how many additional enhancements you can avoid by typing parameters with an ABAP Dictionary type. As mentioned previously, it is because the type `BOOKINGS` is enhanced that all fields of this type get the new component "automatically."

Though we could do with far fewer enhancements because we use the type `BOOKINGS` so extensively, we still have a lot enhancements in our little project. The framework forces us to organize our enhancement in any case, and we are well advised to structure our enhancements in a way that fits the structure of the components and subcomponents they enhance in our projects.

As shown in **Figure 15** on page 125, we group the enhancements for the two components in two different composite enhancement implementations `CEI_VIEW_BOOKINGS` and `CEI_UPDATE_BOOKING`, and choose the composite enhancement implementation `CEI_BOOKINGS` as a container for both. Inside we have different enhancement implementations for the Web Dynpro enhancements, the source code plug-in, and the `overwrite` exit. The two table appends do not belong to any of the Enhancement Framework containers — this is for the simple reason that table appends are not part of the Enhancement Framework. Still, as you can see, they interact perfectly well with the framework.

It might look a bit academic to work with so many containers without a lot of enhancements, but this example is meant to show the principle of how to organize the elements of an enhancements projects. In a real-world project, you will probably have far more enhancements for which containers will prove to be very helpful.

Helpful hints for using the Enhancement Framework

As you have seen, the Enhancement Framework is a powerful technology. Here are some helpful hints and cautions that you need to keep in mind when using this framework to enhance your applications:

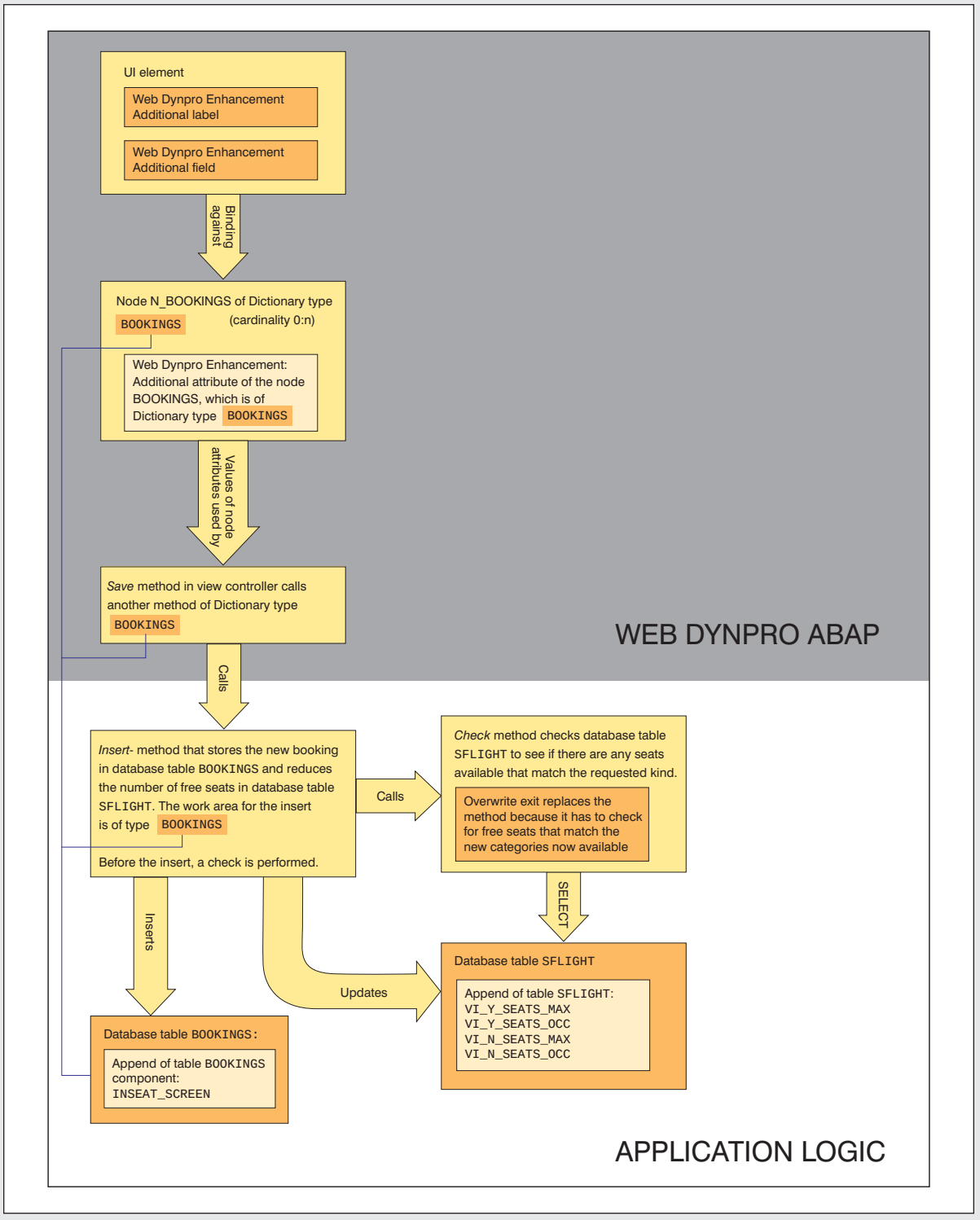


Figure 14 Adding a new field and label to the Web Dynpro view

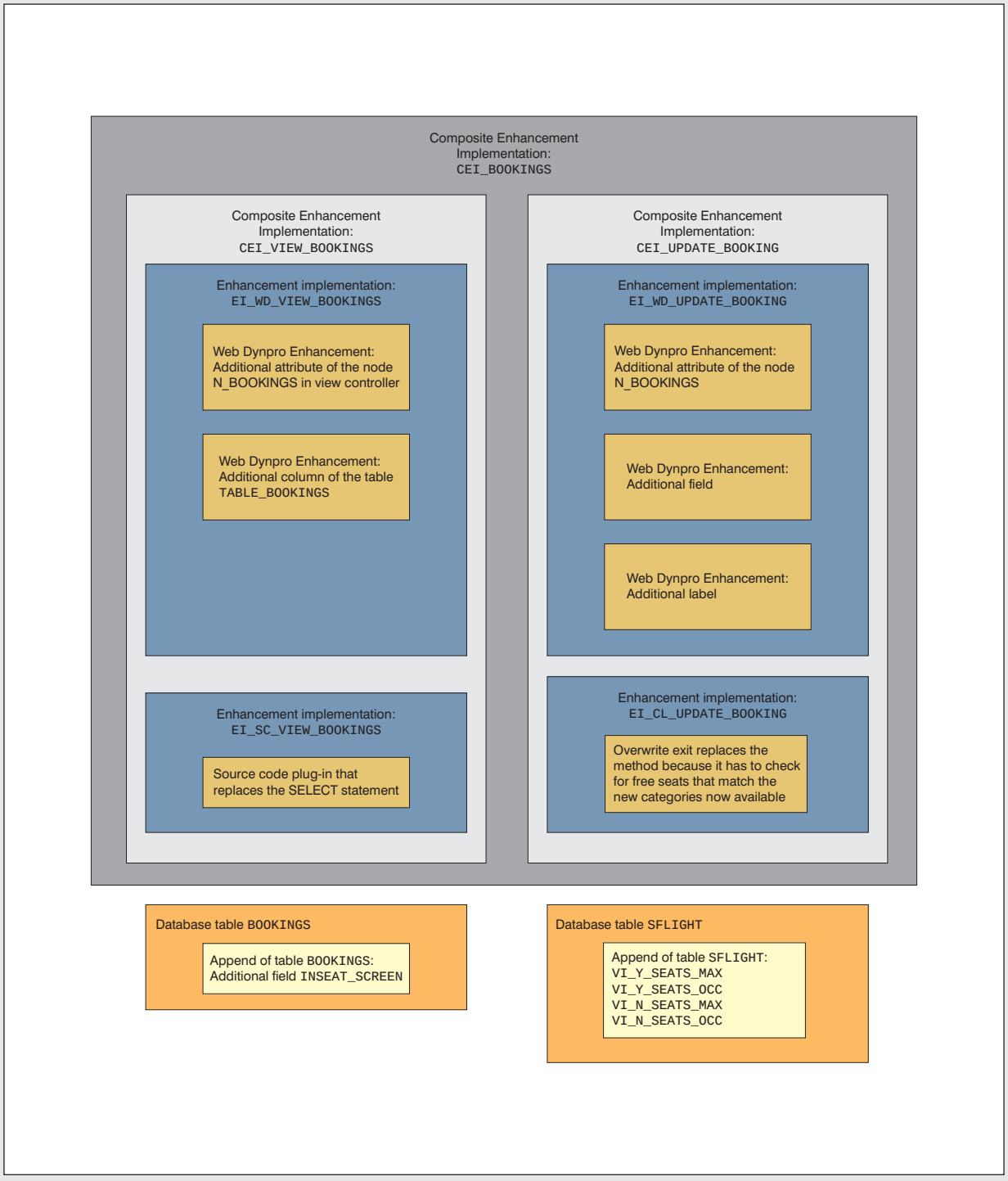


Figure 15 Organizing the example project

- **Know what you're doing and use the new framework with care.** Enhancing an application with this framework is an adaptation at a very low technological level. There are no protection mechanisms to prevent you from typing in coding that may run counter the logic of the original program. Enhancing an application without detailed knowledge of its implementation and underlying business logic not only may lead to a result other than the one you want, but could damage the original application. For example, code intended to secure the validity and integrity of data may no longer be reached if you leave a method or function module irregularly, for example, by issuing a RETURN or throwing an exception in the enhancement.

  Because there is no such thing as a semantic check for your enhancements, it is up to you to take care when designing and implementing an enhancement so that it does not run counter to the underlying business logic of the application you enhance.

- **Plan and organize your enhancements from the very beginning.** An enhancement project usually means adding some new semantic feature to an application and comprises all the changes required to make this work. In general, this necessitates a large number of enhancements across different development objects, as you saw in the example: Remember how many enhancements plus appends were necessary to simply add a column to a table and show it as a field on a screen. It is easy to get lost in a large number of enhancements, so it is a good idea to keep them organized from the very beginning in composite enhancement implementations.

  An enhancement project requires thorough and systematic planning, an expert who is familiar with both the implementation and the logic of the application, and a sound knowledge of the Enhancement Framework with its scope and limitations. It is not advisable at all to hack in an enhancement every here and there.

- **Control the creation of enhancements in your company.** In contrast to modification, customers do not need a modification key to implement an enhancement option. Nevertheless, it is possible to prevent developers with a usual ABAP development role from enhancing SAP programs in a completely uncontrolled way. Enhancement implementations are transport objects of their own type, and this offers a way to technically restrict the creation of implementations. You may create the respective authorization that forbids the creation of objects of type ENHO (enhancement implementations) in your customer system, so that some developers have the authorization to create enhancement implementations, and others do not. In fact, it is strongly recommended that you manage and control the creation of enhancements in your company this way.

- **Know when to enhance an application.** You should change an SAP solution only if you have come to the conclusion that none of the possibilities of customizing meet your specific requirements. First, try out everything that is possible by means of customizing. If you still need to adjust the standard, you should then consider using the Enhancement Framework. As enhancements are objects of their own, they will cause you far less work in an upgrade than modifications. In addition, the Enhancement Framework offers a variety of useful features, so in many cases there may be different ways to achieve your aim.

- **Choose the right enhancement technology:** Though customer enhancements are objects in their own right and belong to the customer`s namespace, the enhancement option they are assigned to may vanish or change in an upgrade. Though the transaction SPAU_ENH offers tool support to adjust the enhancement options after an upgrade or an support package installation You can minimize this effort by choosing an appropriate enhancement technology:

  - High-order enhancements technologies like BAdI or CMOD should be your first choice.

  - If there are none of these technologies around to make the change you need use class enhancements or source code plug-ins in function groups

How classic modification technology and the new Enhancement Framework interact

Though there are plenty of implicit enhancement options around, sometimes you might face a situation in which neither an implicit nor an explicit option meets your requirements for an adaptation of the SAP code. In this case, it is a good strategy to insert the suitable enhancement option you need by means of a modification and then implement this option in the usual way. This way you have your new code in your namespace in an enhancement and you can profit from the strong tool support of the Enhancement Framework.

or reports. How stable a particular enhancement option of this kind
is heavily depends on its particular position: Generally an enhancement option in a function module or a public method is more stable than one in a private method or a form routine. In many cases, an experienced developer can probably assess himself how stable a particular enhancement option probably is.

For an interesting possibility to use modifications in combination with the Enhancement Framework, see the sidebar on the following page.

Conclusion

The basic idea of the enhancement framework is to have a technology that serves the same aim and fulfills the same functions as modifications, without any of the drawbacks that are inherent to the very concept of a modification — a modification is part of the object it modifies, which means you have to reinsert modifications of SAP objects after every upgrade. In other words, modifications cause quite a bit of work. In contrast, enhancements are transportable objects in their own right that you develop within your own namespace. For this very reason, enhancements cause far less work during an upgrade — an enhancement is not part of the object it enhances, which means it is not changed if the object changes. Rather, enhancements are attached to the original object via “hooks” called enhancement options. Once an enhancement option is reached by the control flow, the enhancement attached there is processed and then the control flow returns.

With this new technology you can enhance global classes, function modules, Web Dynpro ABAP components, and all source code units using implicit enhancement options provided by the system. Moreover an application developer can define additional explicit enhancement options for source code plug-ins and new kernel-based BAdIs, which are also integrated in this new framework.

To support a proper organization of your enhancement projects from the outset, all explicit enhancement options and all enhancements have to be part of containers provided by the framework, that is, simple and composite enhancement spots on the definition side and simple and composite enhancement implementations on the implementation side. These containers underscore the importance of careful planning and design in an enhancement project: Even small semantic changes will lead to quite a number of enhancements. Most probably you will end up with far more enhancements than you suppose in your project, so you should stay organized from the very outset.

Keep in mind that the new Enhancement Framework isn't an open invitation to freely enhance your SAP solution in as many ways and means as possible, however. It is a powerful tool — think of it like a high-end precision drill, which will probably not tempt you to sprinkle holes all over your walls and furniture. It is merely a comfortable professional means by which you drill the best hole possible whenever you absolutely need to.

These hints of caution are in no way intended to dissuade you from enhancing a piece of standard software. It is only because it is so easy now with the Enhancement Framework that the users of this framework are better off with some more deliberation.

We hope that this article has provided you with a solid understanding of the conceptual side of this new framework and the new opportunities it brings.

Thomas Weiss has a PhD in analytic philosophy and worked as a professional writer before he joined the SAP NetWeaver product management training team in 2001, where his responsibilities included the e-learning strategy for ABAP. After getting increasingly into writing ABAP material himself, he is now a member of the SAP NetWeaver Application Server Product Management. One of his main interests lies in rolling out ABAP topics both for experts and for beginners by writing weblogs in SDN. You may reach him by email at thomas.weiss@sap.com.

After studying computer science, Michael Acker started working at SAP in 1994. In the following years, he developed innovative applications that automated SAP CD/DVD production and archiving. Since 2001, he has been working as a developer and architect in the Development Workbench (SAP NetWeaver) group. He is applying his skills to the modification and enhancement tools, as well as to the modification adjustment tools. You may reach him by email at michael.acker@sap.com.

