Unveil the power of loosely coupled composite applications by replacing services for additional functionality

by Volker Stiehl

Volker Stiehl
Product Manager,
SAP AG

Volker Stiehl received a degree in computer science from the Friedrich-Alexander-University of Erlangen-Nürnberg, Germany. He was a consultant for distributed J2EE-based business solutions and integration architectures at Siemens for 12 years. In 2004 he joined SAP's product management team for a composite application development toolset, where he's also responsible for methodologies and best practices. He holds workshops in composite applications and is a regular speaker at conferences, such as SAPPHIRE, SAP TechEd, and JavaOne. You may reach him at volker.stiehl@sap.com.

The power of loosely coupled composite applications can be summed up in one word: flexibility. Whether your company upgrades to the latest SAP release, requires new business logic, or needs to integrate new back-end systems as the result of an acquisition, the capability of composite applications to replace enterprise services enables you to change functional implementations as needed for your business. In fact, this flexibility through replacement is one of the key benefits of composite applications.

The replacement technique accelerates the development process by decoupling dependencies between the tasks of various developers. Because you define one common interface for a composite application, you can plug in various types of business logic — different implementations — on the other side of that interface. You can even delegate an implementation to any back-end system in your landscape by mapping the data from your interface to the interface of the back-end service that you need.

I hope you can see the beauty of this technique: You can create a well-defined interface for your composite application and then plug in whatever implementations you need, such as:

- A dummy implementation to speed up development so you don't have to wait for other developers to finish their tasks
- Your self-developed business logic in the form that the composite application requires
- A standard SAP enterprise service that already exists
- Any back-end service you have that fulfills the business need
- A call to SAP NetWeaver Process Integration (SAP NetWeaver PI) to integrate service-enabled legacy systems

These examples demonstrate one of the core benefits of composite applications: flexibility. You can react to almost any change that occurs in

either your IT landscape or your business functionality. Just plug the newly required functionality into your composite application, and you're done! The composite itself remains untouched.

This article adds some important functionality to my earlier three-part series on developing composite applications: "Develop composite applications with SAP NetWeaver Composition Environment 7.1: Part 1 — Enterprise services and their usage within a composite application" (*SAP Professional Journal*, May/June 2008); "Develop composite applications with SAP NetWeaver Composition Environment 7.1: Part 2 — User interfaces and the tools that help you create them" (*SAP Professional Journal*, July/August 2008); and "Develop composite applications with SAP NetWeaver Composition Environment 7.1: Part 3 — Modeling collaborative business processes with SAP tools" (*SAP Professional Journal*, September/October 2008). In this article, I refer to these articles simply as Part 1, Part 2, and Part 3, respectively. This series of articles has shown you how to approach developing a complete composite application from scratch and has given you insights into the overall development process for applications sitting on top of an enterprise service-oriented architecture (enterprise SOA) landscape. I assume that you have read the articles in that series; this article builds upon that knowledge.

This article also contains a tip that is useful in developing SAP Guided Procedures (SAP GPs), which were discussed in Part 3. This tip, shown in the sidebar on the next page, explains how to monitor processes while they are running and how to analyze issues during process execution.

Replacing enterprise services

In Part 1, I showed you how to provide a simplified interface for a complex enterprise service and how to create a dummy implementation for that interface to give it as much flexibility as possible. This article shows you how to replace that dummy service implementation for the GetCustomerBasicData call with an actual enterprise service, demonstrating the real value of back-end abstraction in action. As you know, all three user interfaces (UIs, detailed in Part 2) use the GetCustomerBasicData call. As a consequence, *all* UIs benefit immediately from this replacement without needing to explicitly adapt their models. This may sound like a miracle, but in fact, you just benefit from applying good composite application best practices.

You may be wondering why you would want to replace your implementation. Here are a few potential use cases:

- **Start with a dummy service implementation and then replace it with the real-world service:** This is the use case I showed in the example in Part 1. Often, complex services just aren't ready for consumption. However, you already know the service interface, so why wait for the service implementation? To continue with development on the consumer side from either the UI (Part 2) or the process layer (Part 3), provide some sample data as part of a dummy implementation and then replace it afterward. When you do this, you get the best productivity from your developers because you have reduced the amount of time they spend waiting for each other.
- **If you're not satisfied with an implementation or your business requirements have changed:** You figure out that a given default implementation of a service doesn't fit your needs in some aspects. In this case, you want to enhance the service's functionality. Once the new implementation is available, you want to be able to switch to it easily.
- **If your landscape beneath the composites changes because of a release update or a merger or acquisition:** This is a common use case. Since composite applications work on top of existing applications, the probability of a release change in one of those systems is very high. In addition, the interfaces of the existing applications' APIs often change as well. You're in a situation where the consuming layers don't explicitly use those interfaces because you've wrapped the consuming layers in your own interface. So it's easy to create new implementations connecting to the new APIs without disrupting the consuming side.

Monitoring SAP Guided Procedures

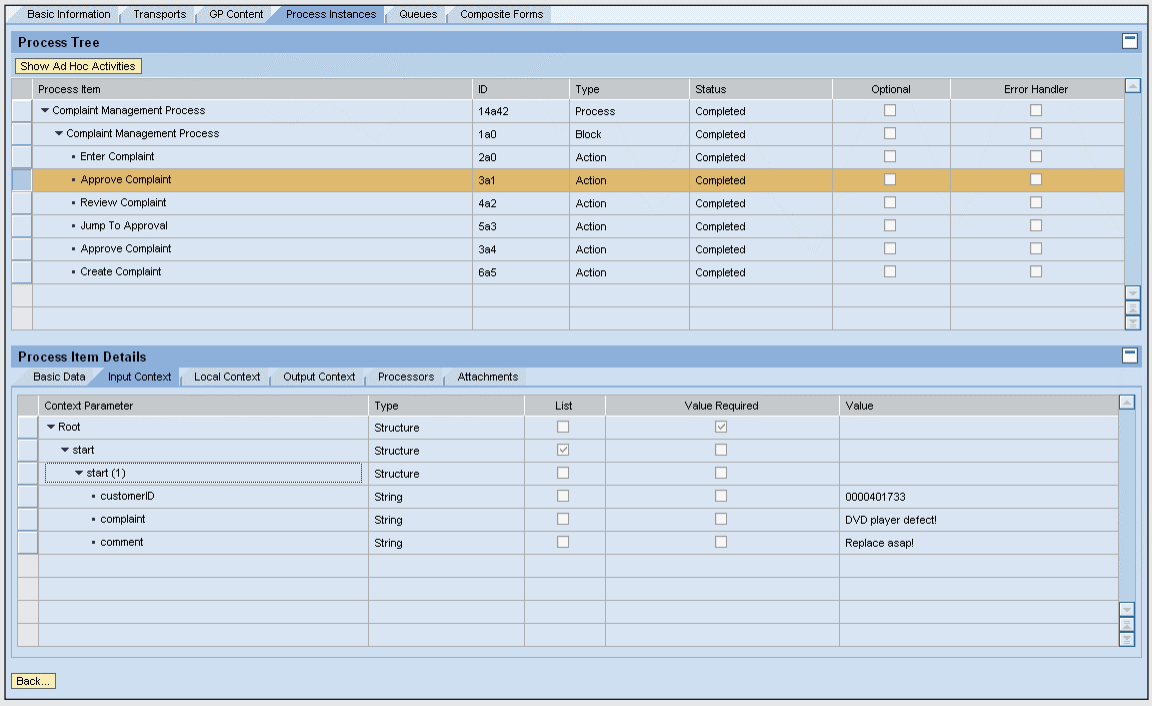
Imagine that you've started executing an SAP GP and you find that a particular work item doesn't appear on your work list. How can you determine in which step the error occurred and why? An SAP GP comes with a tool for monitoring your processes independent of their state (e.g., erroneous, still running, terminated). It's part of SAP NetWeaver Administrator.

To find out where the error occurred, call SAP NetWeaver Administrator via http://<*host*>:<*port*>/nwa, and navigate to Availability and Performance Management → Process Monitoring → Guided Procedures.

This page has several tab strips, the most interesting of which, in this example, is the Process Instances tab strip shown in the first figure. Navigate to this tab and search for either Completed (if your process finished successfully) or Erroneous instances. Then, add the user and account with which you instantiated the process to the Initiator field and click the Go button.

Select the process instance that was found and click the Process Items button. Now you should see your process with all of its completed actions in the second figure.

Next, select the entry in the process instance list that appears. A details section about the selected process is also shown below the list. Now, click the Process Items button.



Then, select one completed action and browse through the various tab strips to get a feel for the data captured during runtime. In the figure, the Approve Complaint step is selected together with the data being entrusted to the step (the figure shows the Input Context tab). If you click the Output Context tab, you see the data that the step returned to the SAP GP context. By using this monitoring tool, you can get a fairly good understanding of how the process executed — in what sequence — and how the data flows during runtime.

A word of caution:

As the figure shows, the SAP GP framework records three contexts: input, local, and output. This data is all stored in the database. However, consider what happens if the SAP GP's context memory becomes too large, for example, 1GB. (I'm talking about an *average* size of 1GB for all nodes; note that every node [process, block, action] has its individual context size depending on the modeled parameters.) For every action in the Process Tree section of the second figure, SAP GP captures the three contexts; it also captures them for the block and the process (see column Type in the top part of the figure).

The developer should keep an eye on the SAP GP context because the disk space needed for the context data can easily explode and generate quite a lot of data in the associated database. To calculate the amount of space you should take into consideration for the context space, you would have to multiply the context size by the number of executed process, block, and action nodes (in this case, 8); then you multiply this total by 3 for the three contexts. Altogether, you get a final result of (8 nodes * 3 contexts * 1GB) = 24GB of space allocated to get 1GB per SAP GP context. This formula should help you roughly estimate the required disk space for each process instance.

You can imagine how the size of the context data can increase if you think of really large processes with lots of data being stored in the SAP GP context. I recommend storing only the primary keys for master data in the SAP GP context and rereading the data in the UIs if necessary. I also recommend splitting large processes into several smaller chunks so that you can reduce the nesting of blocks within blocks to a minimum. In addition, I advise that you use 15 actions as an appropriate number of steps per process. Make sure you check out the "Usage guide for creating Guided Procedures" on SDN.[1]

[1] https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/7308

- **If you move your application to a different department or region or you want to sell your composite to a new customer:** These two scenarios have one thing in common: You can't expect to find the same system landscape in the new location that you initially used to build the composite. It is probable that the new landscape — along with the service APIs — will differ significantly and you will have to wrap them according to your composite application's needs. Once again, the technique of back-end abstraction comes to the rescue.

In all of these cases, you need exactly the kind of flexibility that replacement gives you and that

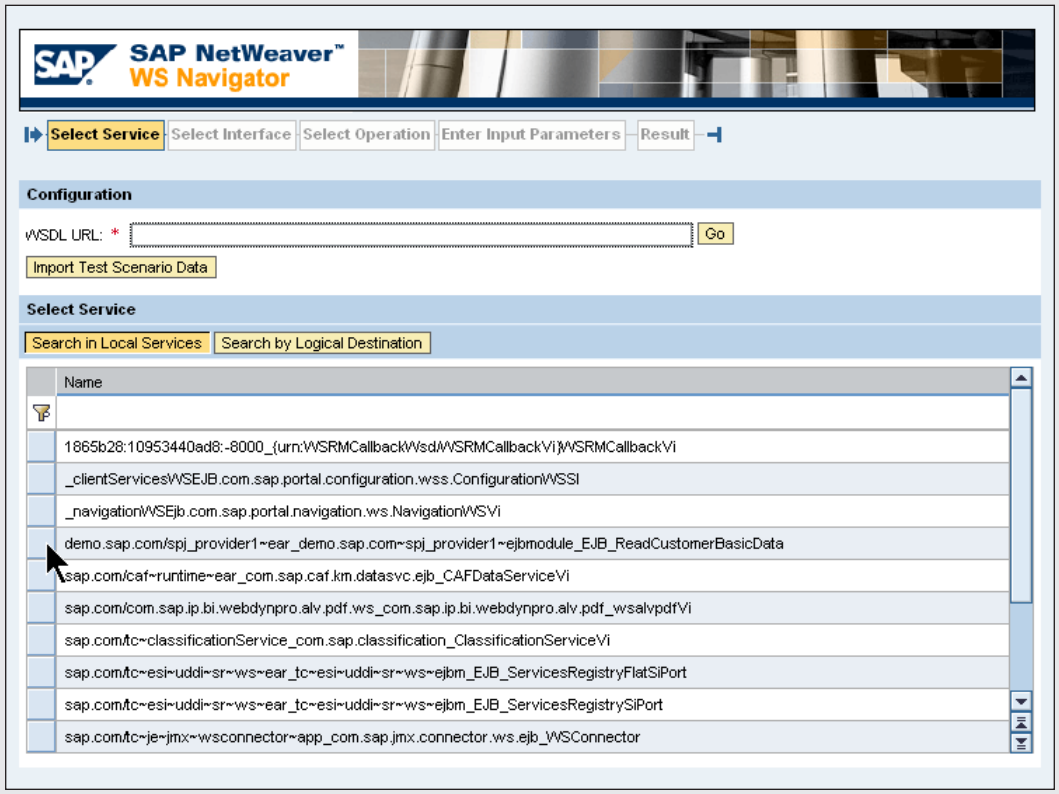


Figure 1 Searching for the enterprise service you need to replace

you will lose if you connect applications by hard-wiring them.

How to achieve application replacement

Step 1. Create a new project in the SAP Composite Application Framework (SAP CAF) and name it spj_provider2. This project will contain the new service that will replace the dummy implementation I provided in the project named spj_provider1 (in Part 1).

Step 2. Create a new code skeleton for the interface you'd like to replace. This step is crucial. Since you want to replace a service implementation, it must have *exactly* the same interface based on the Web Services Description Language (WSDL) as the original service implementation.

1. To determine which interface was the original one and where to find the WSDL, recall Part 1. You published the original interface as a Web service in Part 1. Because you need the URL of the WSDL file (it's used as a blueprint for the new service), open the Web Service Navigator and select the ReadCustomerBasicData service, as shown in **Figure 1**.

2. Once you have selected the service, you will see the URL for the service's WSDL file on the next screen (see **Figure 2** on the next page). Copy the URL (you'll need it in SAP CAF).

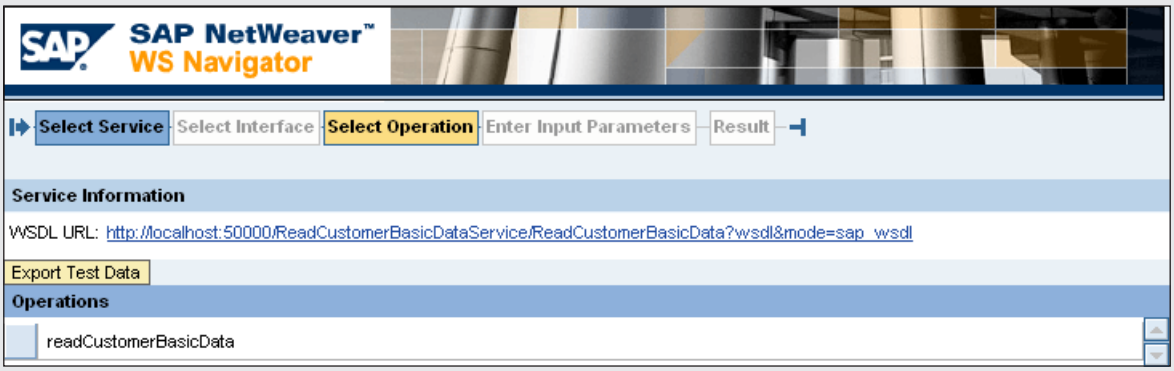


Figure 2 The URL of the original service's WSDL

3. In SAP CAF, right-click the external node of your newly created project and select the Create Web Service Provider option from its context menu, as shown in **Figure 3**.
4. The WSDL importer starts and asks for the source URL of the WSDL file. Since you want to provide the WSDL URL directly, select the Remote Location/File System radio button, as shown in **Figure 4**, and proceed to the next screen.
5. Paste the copied URL into the input field and click Finish, as shown in **Figure 5**.
6. Behind the scenes, SAP CAF retrieves the WSDL file and automatically creates a new application service that has exactly the same interface as the original service that you implemented in spj_provider1. At the same time, SAP CAF makes the application service available as a Web service as well. **Figure 6** on page 26 shows you the result of this step. Note that the interface in project spj_provider2 is identical to the interface in project spj_provider1.

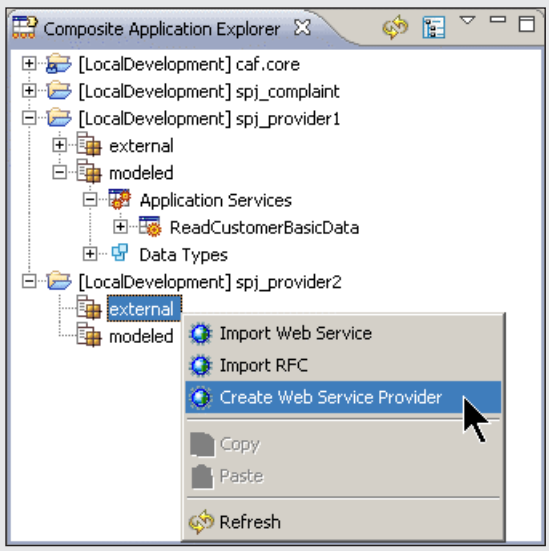


Figure 3 Creating a Web service provider in SAP CAF

Step 3. Import the enterprise service you want to call. The goal is to delegate the service call to an external enterprise service. To do this, you need its service interface. Import the appropriate service CustomerBasicDataByIDQueryResponse_In by right-clicking the external node and choosing the Import Web Service option from the context menu. (If you need a detailed description of how to do this, see the section "Step 2: Simplify service interfaces with CAF" in Part 1.)

Step 4. Map the simplified interface of your application service to the complex interface of the external enterprise service. To achieve this, you have to open the newly created application service

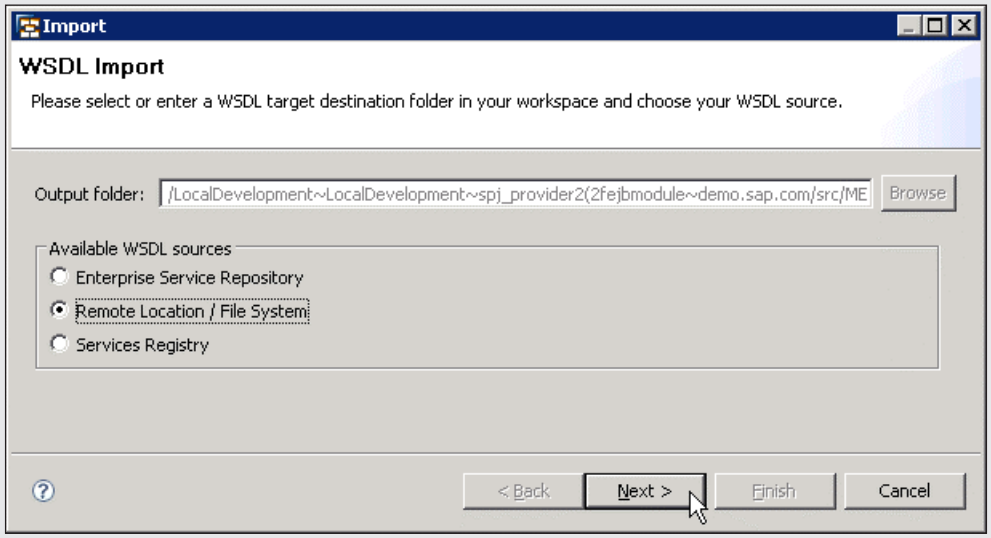


Figure 4 Importing the target WSDL

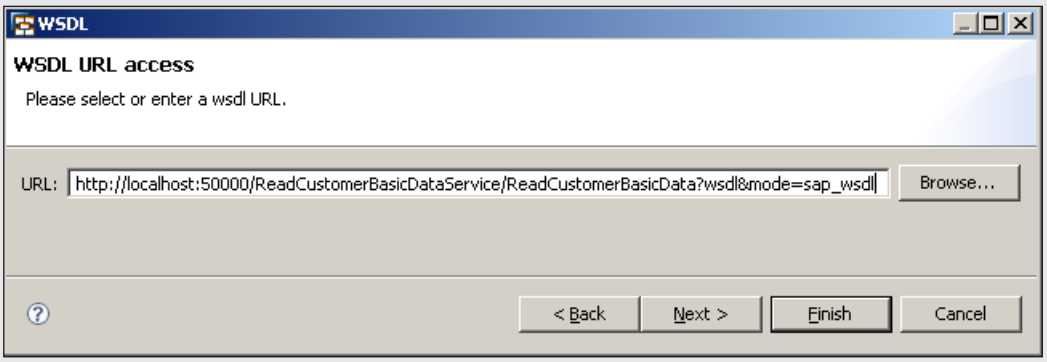


Figure 5 Entering the URL for the original WSDL file

ReadCustomerBasicData. Since this application service was automatically generated, it isn't placed under the modeled node, as you might expect. You'll find it under the external node, as shown in **Figure 6** on the next page.

1. Open the ReadCustomerBasicData application service and select the Operations tab strip. You'll find that your operation is already defined, but the checkbox in the Implemented column is set to Yes. This means that you want to implement the functionality yourself via Java programming. In this case, you should delegate the call, not implement it.

2. Uncheck the checkbox and go to the Datasource tab strip, which maps interfaces to each other (the Source operation hasn't been assigned yet).

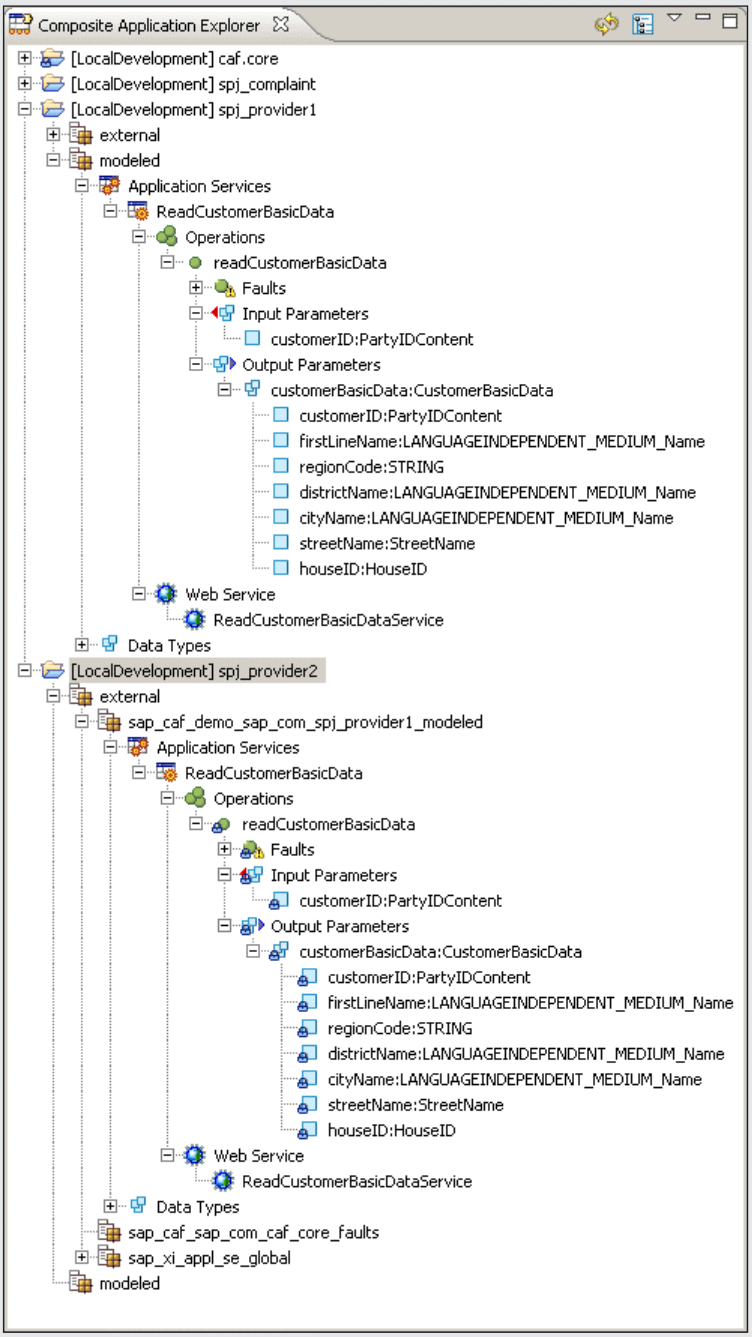


Figure 6 Results after creating the Web service provider

Click the Create mapping button to open a dialog for choosing the enterprise service that you previously imported in Step 3 (Import the enterprise service you want to call). **Figure 7**

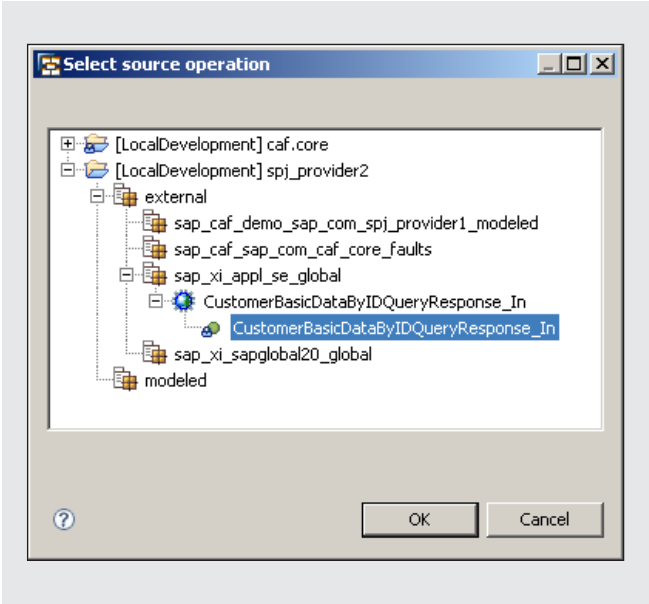


Figure 7 Selecting the source operation

shows you where to find the imported service. (This is really a very cool feature, almost a Web service proxy over an existing Web service.)

3. Select the service and click OK. Now you have both interfaces side by side: the simplified interface of the application service and the complex interface of the external enterprise service, as shown in **Figure 8**. What's left is a mapping on the field level: You need to determine which field to transfer from the simplified interface to the complex one, and vice versa, and how you can model it within SAP CAF.

4. Look at the mapping area you find on the ReadCustomerBasicData tab. It's separated into two parts: on the left side, you have the Target operation containing all of the fields in your simplified interface; on the right side, you identify the Source operation containing all of the fields in

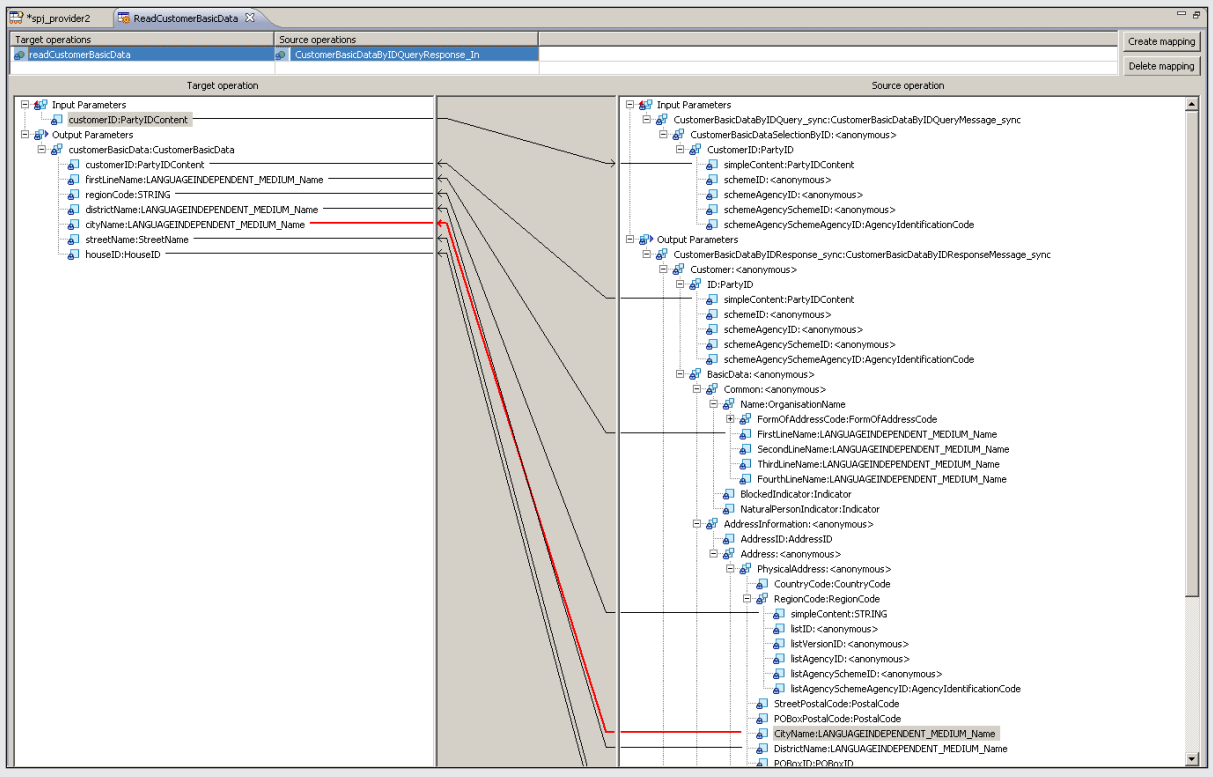


Figure 8 The table of operations

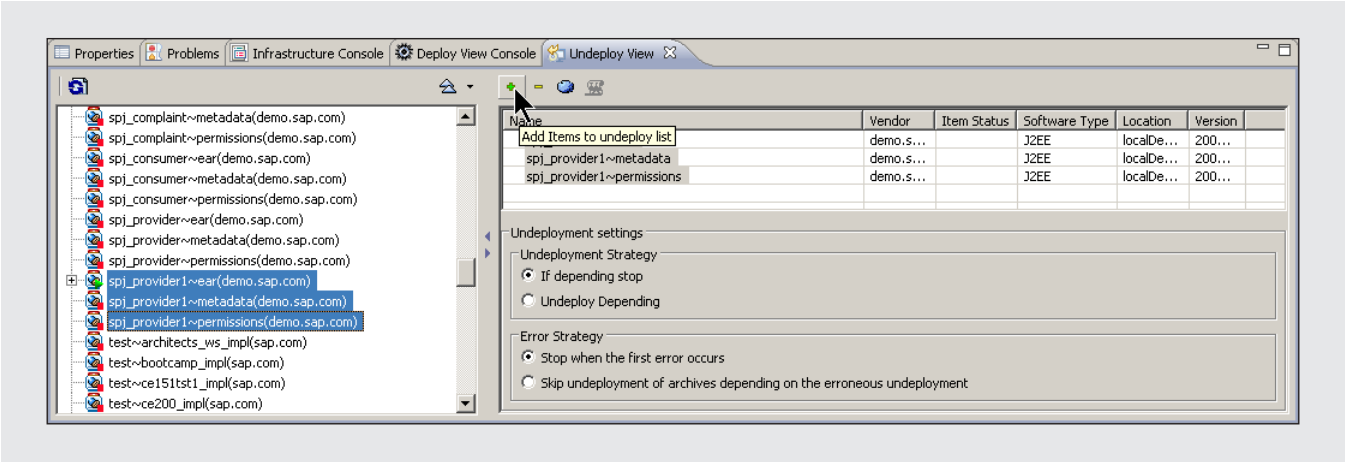


Figure 9 Undeploying applications

the complex interface. Drag and drop those fields that you need to map from left to right for input fields or from right to left for output fields. The final result should look like **Figure 8**.

The mapping works very smoothly because you used the same data types for the simplified interface as you did for the complex interface. This was one of the recommendations I gave in my previous articles, and now you can see how nicely everything fits together. Keep in mind that with the current version of SAP CAF, it's not possible to map fields of different data types to each other. This change will come in a forthcoming release of SAP NetWeaver Composition Environment (SAP NetWeaver CE).

Step 5. Undeploy the old implementation, and deploy your new application service. Since you've created, more or less, a twin of the original service with the same WSDL description, you can't deploy the new service directly because it will conflict with an already running Web service (your first implementation). However, the SAP NetWeaver Developer Studio (NWDS) comes with a new Eclipse view called Undeploy View. You can open it from the menu; select Window → Show View → Other… and choose the Undeploy View that you find under the Deploy View node.

Step 6. Use the Undeploy View to connect to your server instance and retrieve all deployed applications. From that list, you can choose the applications that you'd like to remove from the server. (You have to enter your administrator account to perform this step.)

1. Select the three applications that are in the namespace spj_provider1 and click Add Items to undeploy list button ( ), as shown near the top of the screen in **Figure 9**).

2. Once the applications appear on the list, click the button Undeploy all items in the list ( ), which is also near the top of the screen, to actually remove them from the server. You'll get a message about successful execution. Now you're ready to generate, build, and deploy your new implementation.

Step 7. Provide a logical destination for your enterprise service. You need to connect the application service that you just created to the external service, which contains the real business functionality. Within NWDS, you only imported the service interface's *description* to work with the fields. However, this doesn't actually connect the application service to the interface.

Therefore, you need to create a logical destination, a function of SAP NetWeaver Administrator that is delivered with SAP NetWeaver CE. Basically, you provide a name for your Web service and the service consumer works only with this name, not directly

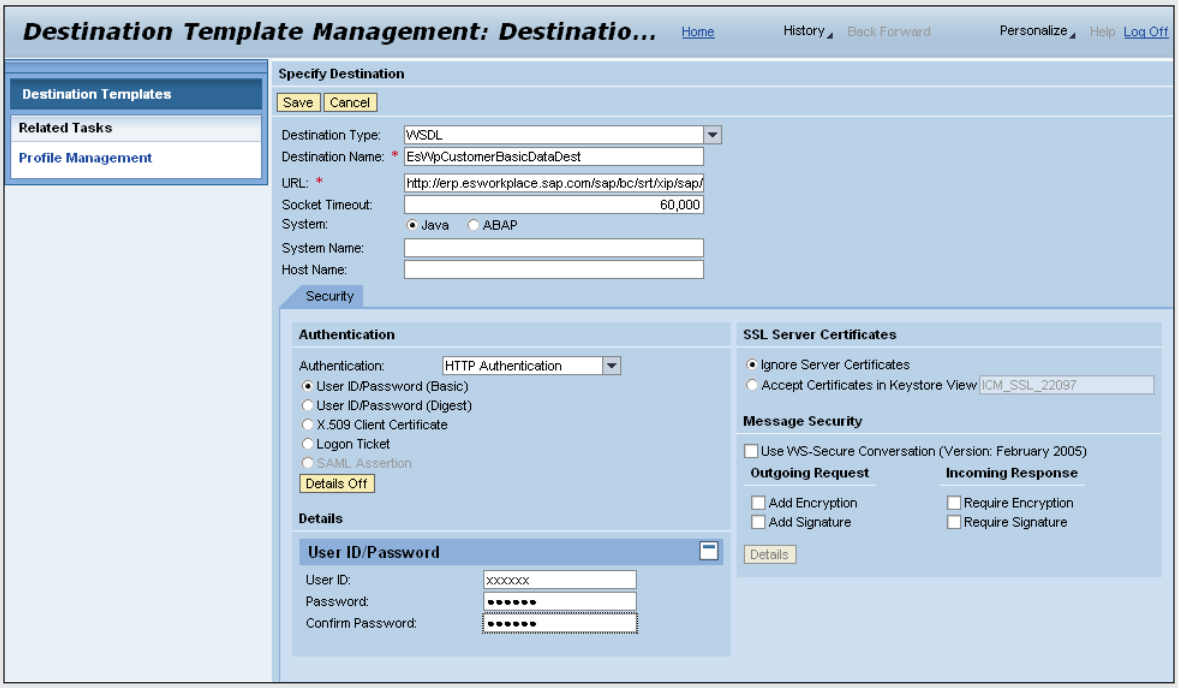


Figure 10 Setting a logical destination for the enterprise service

with the Web service. To create a logical destination, follow these steps:

1. Open SAP NetWeaver Administrator via URL http://<*host*>:<*port*>/nwa.
2. Click SOA Management → Technical Configuration.
3. Click Destination Template Management to get to the editor for logical destinations. A table lists all previously defined destinations.
4. Click on the Create Destination button to generate a new destination.
5. Fill in the fields as follows:
   a. Destination Type: WSDL (choose from drop-down list)
   b. Destination Name: Enter the name EsWpCustomerBasicDataDest
   c. URL: Use the URL of the WSDL file for CustomerBasicDataByIDQueryResponse_In. You can find it in the Services Registry of the Enterprise Services Workplace (ES Workplace). The details for retrieving the URL are in Part 1. Take the URL from the Endpoints tab in the Services Registry, not the General tab.
   d. Authentication: HTTP Authentication (choose from drop-down list)
   e. Activate radio button User ID/Password (Basic).
   f. Click the Details button so that you can add the credentials for accessing the back-end system of the ES Workplace on which the enterprise service actually runs.
   g. Enter your credentials in the respective fields: User ID, Password, and Confirm Password.
6. Save your entry and see the final result, as shown in **Figure 10**.

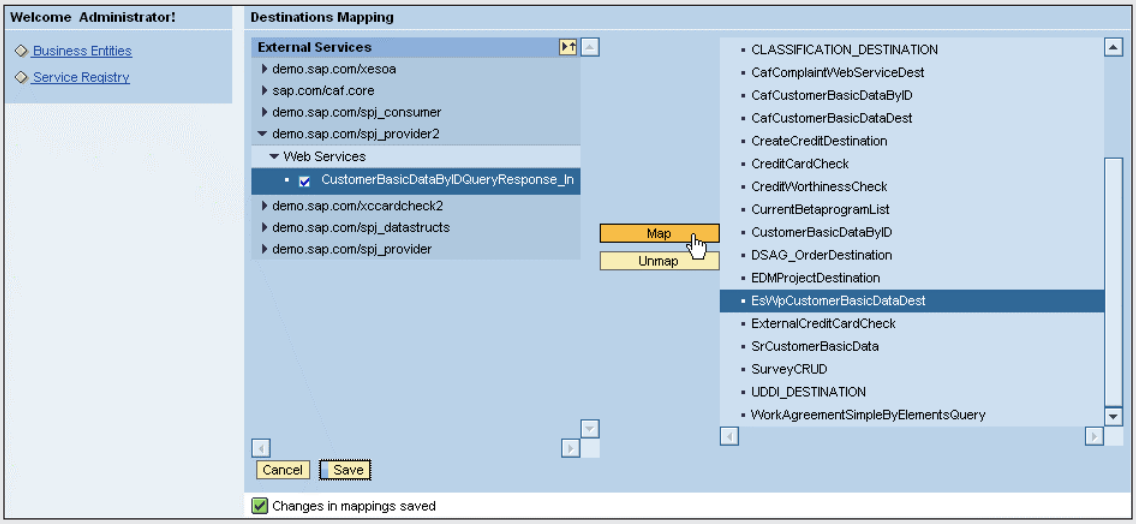


Figure 11 Connecting the application service and the implementation

Step 8. Connect the application service with the external service. Now that you have defined the logical destination, you can connect the application service to the external service. SAP CAF comes with a configuration tool to do this.

1. Call http://<*host*>:<*port*>/caf in your browser (it opens the SAP CAF home page) and navigate to Administrative tools → External Service Configuration. Click the Service Registry link on the left side of your screen, as shown in **Figure 11**, to get a list of all SAP CAF projects that require external destinations.

2. Expand the spj_provider2 node, drill further down to Web Services, and select the node CustomerBasicDataByIDQueryResponse_In. On the right side, entitled Destinations, expand the Web Services node. You will see all of the logical destinations defined on your server.

3. Look for the destination that you created in the previous step, EsWpCustomerBasicDataDest. Select this node as well. Now that both nodes are selected, the screen activates the Map button. Click it to create the connection between the service requirement of your application service and the actual implementation. You should see a checkmark in front of your selected service. Finally, click Save to store the mapping.

4. Start the Complaint Management Process (details about starting processes can be found in Part 3). Now that you have changed the implementation, you should see detailed data for the selected user retrieved from the back-end system. Try to execute all three screens to verify that no adaptations are necessary to apply the change to them immediately. The following screenshots document the changes on the first two screens of the Complaint Management Process: Enter Complaint, as shown in **Figure 12**, and Approve Complaint, as shown in **Figure 13** on page 32.

Now you've experienced the real power of composite applications in action. You've seen how easy it is to replace implementations and how consumers such as the SAP NetWeaver Visual Composer UIs can immediately benefit from that change without affecting their models. This holds true for other consumers as well, as long as they use the simplified interface.

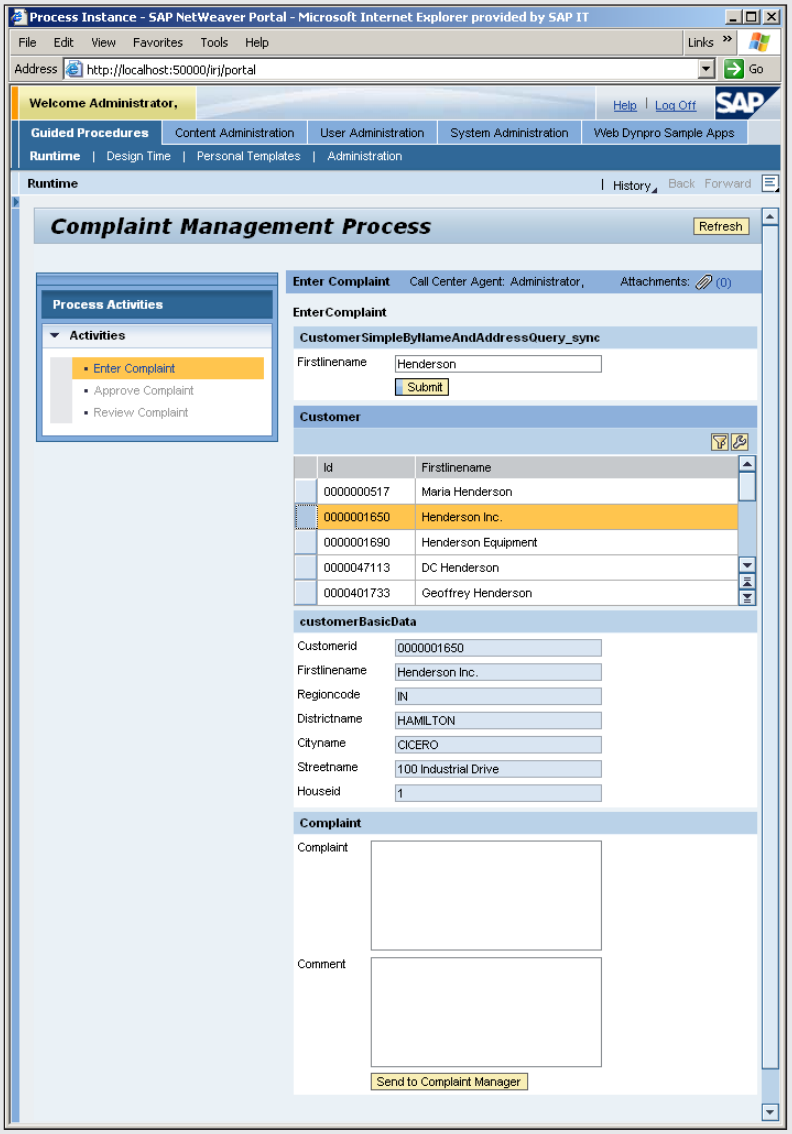


Figure 12 Retrieving detailed data from the back-end system

Conclusion

This article completes my discussion of the topics I introduced in my three-part series on the development of composite applications. It explains in detail how to replace services without affecting their consumers and, thus, shows you for the first time the benefits of loosely coupled applications. If you employ this technique, you will be equipped to face the challenges of changing business requirements or IT landscapes. It is extremely important that you understand the replacement technique so you can apply it to your own development projects. Once you get used to it, you'll never want to be without it.

In addition, this article briefly touches on the monitoring capabilities of SAP GPs. With the Web-based

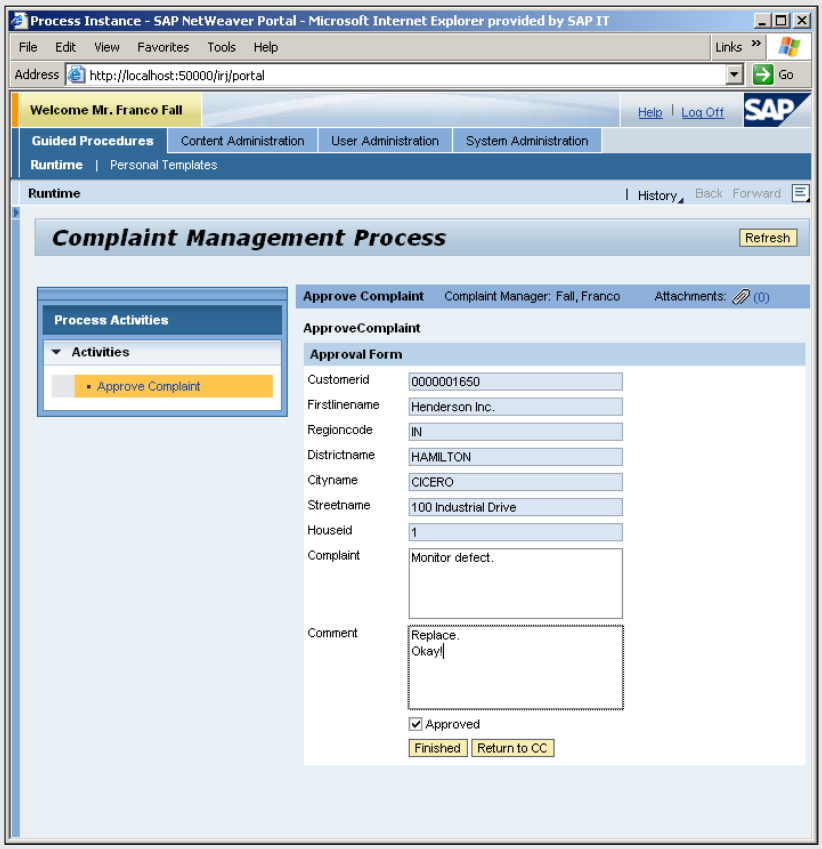


Figure 13 Retrieving data from the back-end system for the account manager screen

tool that resides in the SAP NetWeaver Administrator, you can now analyze the process flow of running, erroneous, or even terminated processes and determine how the data flow looks and where the process bottlenecks are. It's an ideal tool for tracking down errors.

With that, the journey ends. But stay tuned: A new SAP NetWeaver CE release is coming with exciting new features and improvements. SAP is pushing the development experience for composite applications to a new level, and I'm already eager to explain its concepts to you.

