SAP NetWeaver Process Integration 7.1
ES Repository – Service Interfaces

SAP NetWeaver Regional Implementation Group
SAP NetWeaver Product Management
December 2007

THE BEST-RUN BUSINESSES RUN SAP™

Benefits

After reading this document, you will be able to:

- **Understand what a Service Interface is, how it has evolved from the Message Interface, and the main deltas between the two**
- **Understand what an Interface Pattern is, the different types available, and which are relevant for PI usage**

Agenda

1. Service Interface Overview
2. Interface Pattern

Agenda

1. **Service Interface Overview**
2. **Interface Pattern**

Service Interface – Background

Transition

- **Integration Repository → ES Repository**
- **Message Interface → Service Interface**

Similar guiding principles still apply…

- **Outside-in interface design approach still applies**
- **Interface *Category* – Inbound, Outbound, Abstract**
- **Interface *Mode* – Asynchronous, Synchronous**
- **Design Data Type, Message Type → Service Interface**

Main Deltas

- **Service Interfaces may contain several *Operations***
  - **Each Operation describes one communication (synchronous or asynchronous)**
- ***Interface Patterns* and *Operation Patterns***
- **Matching Service Interfaces**
- **Changes driven by enterprise SOA initiatives (but not all changes relevant for PI usage)**

- With SAP NetWeaver 2004, only the Integration Repository existed. The transition to the Enterprise Services Repository began with SAP NetWeaver 2004s, but still without the term “Service Interfaces”, just “Message Interfaces”. Now, with a Service Oriented Architecture as a base and SAP’s drive toward enterprise SOA, with the “next major NetWeaver release”, the Integration Repository and the Message Interface has fully transitioned and evolved into the Enterprise Services Repository and Service Interface.

- Many of the guiding principles from the previous NetWeaver releases were also carried forward to this next major NetWeaver release and still apply. For example, the *outside-in* design approach still holds and many of the service interface attributes such as the category, mode, and data type/message type still remain the building blocks of the service interface.

- The main deltas or new aspects of the *service* interface, even including its name, are primarily driven by the requirements of the enterprise SOA initiative. Service Interfaces, at least as it relates to Exchange Infrastructure (XI), fundamentally, has not changed much at all.

Service Interface – Overview

Service Interface

- **Platform and language independent**
- **Describe operations to be implemented (e.g. leveraging proxy generation) in an application system**
- **Used in mediated scenarios via Integration Server or Direct Connection (p2p scenarios) via Web Service Runtime**
- **Constructed with:**
  - **Message Types, Data Types, and Fault Message Types (optional)**
  - **RFC or IDoc metadata**
  - **External Definitions (e.g. WSDL, XSD, DTD)**

Attributes

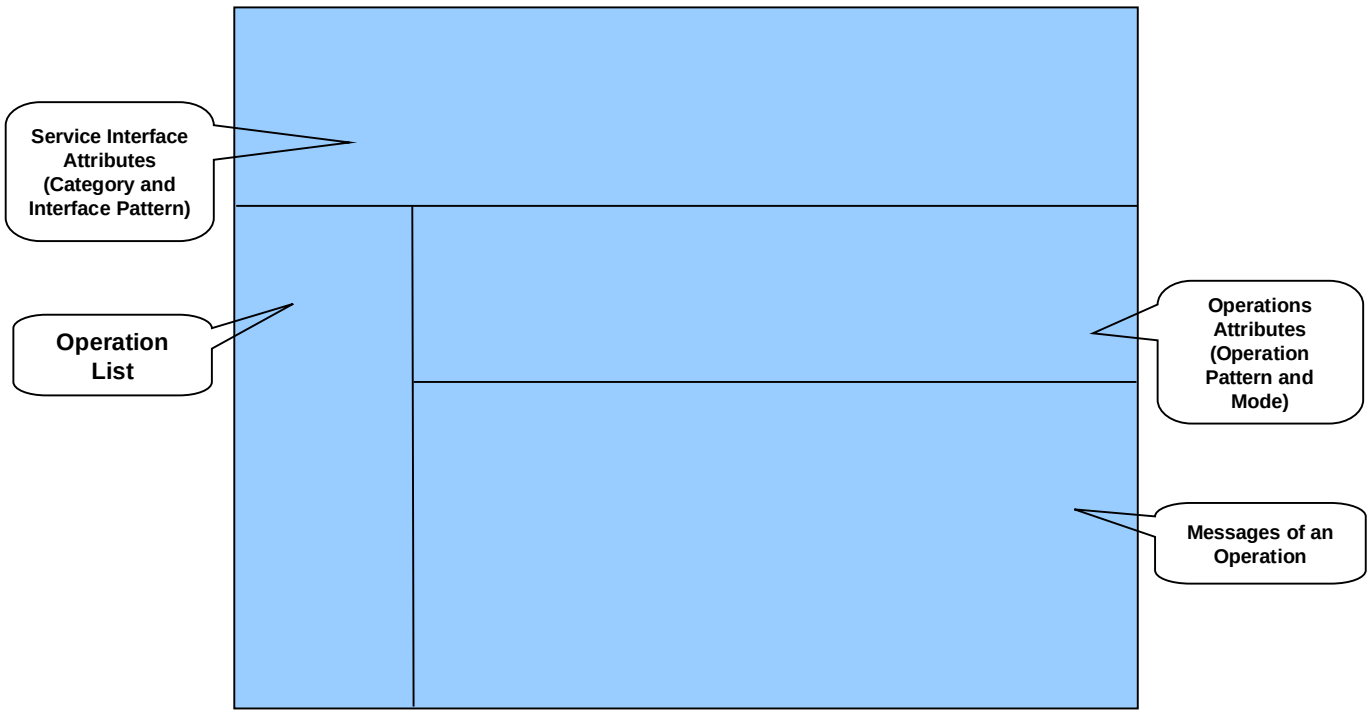
- **Category - *Inbound* (Provider), *Outbound* (Consumer), *Abstract* (Enhanced/Mediated communication using Integration Process)**
  - **Can only have interface patterns Stateless and Stateless (XI 3.0 compatible); for inbound/outbound, all interface patterns apply**
- **Mode (synchronous, asynchronous)**
- **Interface Patterns (stateless, stateless (XI 3.0-compatible), stateful, tu&c/c)**
- **Operation Patterns (depends on interface pattern; e.g. Normal, Confirm, etc.)**

- The Service Interface remains platform and language independent and basically describes a single operation or multiple operations to be implemented in an application system. SAP application systems can leverage proxy generation to facilitate the implementation. The Service Interface can be used in mediated scenarios with the Integration Server and direct point-to-point (p2p) scenarios via the Web Service Runtime. Each operation of a Service Interface is linked to a description of the message structure in the form of an ES Repository Message Type, RFC or IDoc metadata, or an external format in the form of a WSDL, XSD, or DTD.

- The general attributes of a service interface still include the *category* (e.g. inbound/outbound/abstract) and *mode* (synchronous, asynchronous). Inbound and outbound are still terms from the application perspective. As such, and now with a more service-oriented view, inbound interfaces are portrayed with the provider role and outbound interfaces with the consumer role.

- Probably the key change with the next major NetWeaver release is a new attribute called *interface pattern* which will significantly dictate how a Service Interface is implemented on the back-end. The chosen interface pattern drives the available *operation patterns* options that are available. More details on Interface Pattern will be discussed in later slides.

Service Interface – UI Layout

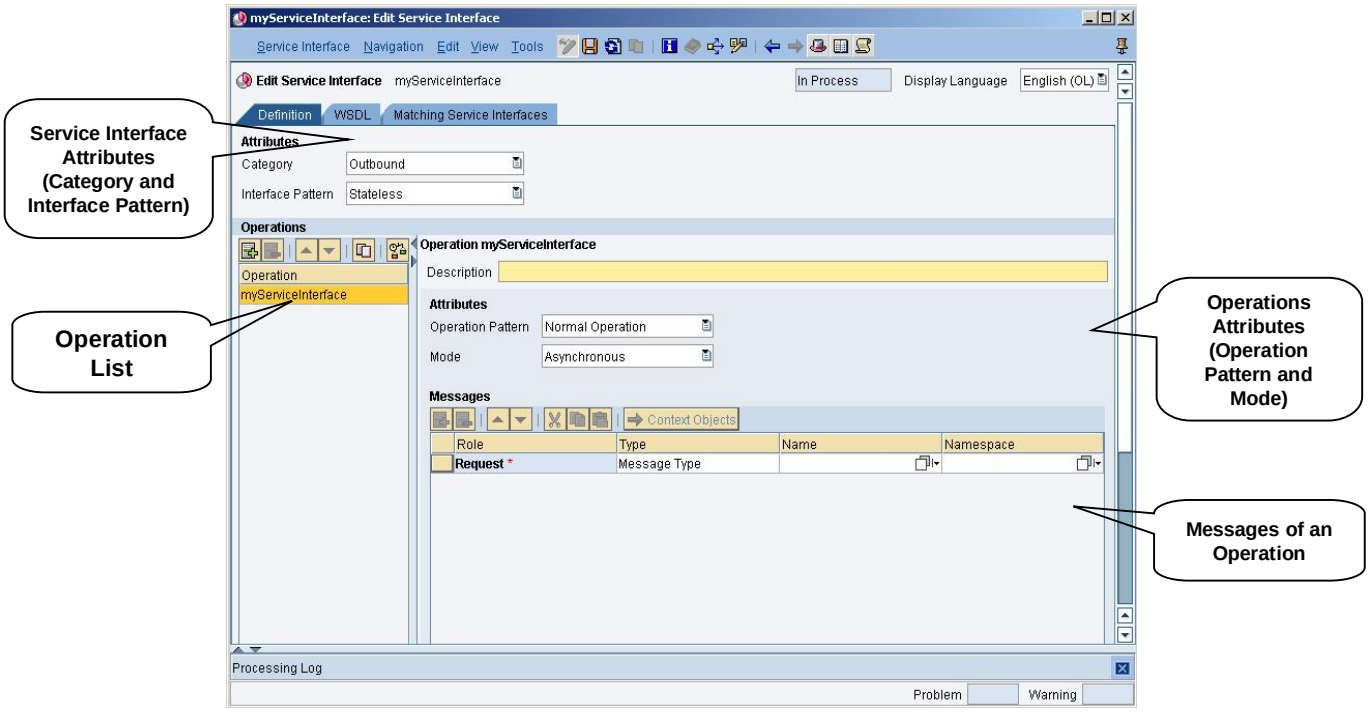


- The Service Interface design screen within the ES Repository has undergone some changes as well. Here, looking first at a plain template screen, you can see four major areas that make-up or define a service interface:

- The main service interface attributes (category and interface pattern) along with the a given operation will determine the operation attributes (operation pattern and selectable mode) and the message associated with a particular operation.

- Also, an example screenshot of the actual UI for the Service Interface design screen for the actual look and feel…

Service Interface – UI Layout



Agenda

1. Service Interface Overview
2. Interface Pattern

Interface Pattern

Communication Terms

- ***Stateless* communication**
- ***Stateful* communication**

Terms relate to state at the *provider* and not the Integration Server

Not to be confused with stateful Integration Processes (e.g. using message correlation)

Interface Patterns using XI Runtime and XI Adapter

- **Stateless (XI 3.0 Compatible)**

Interface Patterns using Web Service Runtime

- **Stateless (via WS adapter in mediated scenarios)**
- **Stateful (mediated scenarios not supported)**
- **TU&C/C (via WS adapter in mediated scenarios)**

Note: Interface Pattern is tightly coupled to the application communication

- Each service interface must be assigned an interface pattern. The interface pattern basically describes the type of communication that is to be executed on the message.
- When referring to **stateless communication**, this is a type of communication in which the messaging runtime does **not** support the saving of a status of a message at the provider once the messaging runtime has completed the message exchange successfully.
- In contrast, when we refer to **stateful communication**, this is a type of communication in which the messaging runtime does support the saving of a status at the provider once the messaging runtime has completed the message exchange successfully.
- Note that the messaging runtime can explicitly support or not support such a procedure. If the messaging runtime supports stateful communication, the application programmer can use the corresponding methods of the messaging runtime.
- Furthermore, when talking about *interface patterns* of a service interface, these terms (stateless and stateful) relate to the state at the provider and not on the Integration Server (in the case of enhanced communication). Therefore, when selecting a stateless interface pattern you can still enhance the communication using the Integration Server to include the execution of a stateful integration process that permits message correlation. So this is to say that these terms are not to be confused when the Integration Server is involved in mediated communication, especially when a stateless interface pattern and a stateful integration process (e.g. using message correlation) is used in the same scenario.
- In terms of how an interface pattern is related to which runtime is used, the stateless (XI 3.0-compatible) pattern uses the XI runtime via the XI adapter and the rest of the interface patterns (stateless, stateful, and tu&c/c) use the web service runtime (and WS adapter in mediated scenarios). Note that the stateful interface pattern is never supported in mediated scenarios via the Integration Server.
- Finally, the selected interface pattern determines how an application developer programs communication in the back end, thus the interface pattern is tightly coupled to the application communication. If you change the interface pattern, the application program in the back end must also be changed. For example, this applies if the interface pattern is changed from *Stateless (XI 3.0-Compatible)* to *Stateless*, and for any other interface pattern change.

Interface Patterns and Operation Patterns

Each Interface Pattern has specific Operation Pattern(s) and Mode(s)

Interface Pattern	Operation Pattern	Mode
Stateless	**Normal Operation**	**Synchronous or Asynchronous**
Stateless (XI 3.0 compatible)	**Normal Operation**	**Synchronous or Asynchronous**
Stateful	**Normal Operation**	**Synchronous**
	Commit Operation	**Synchronous**
	Rollback Operation	**Synchronous**
TU&C/C	**Normal Operation**	**Synchronous or Asynchronous**
	Tentative-Update Operation	**Synchronous**
	Confirm Operation	**Asynchronous**
	Compensate Operation	**Asynchronous**

- This table summarizes the relationship between interface pattern, operation pattern, and mode.

Interface Pattern – Stateless & Stateless (XI 3.0-compatible)

Stateless

- **Single or multiple operations can be defined**
- **WS Runtime via WS adapter**
- **Point-to-Point communication using web services**

Stateless (XI 3.0-compatible)

- **Message Interfaces from NetWeaver 2004s are migrated to service interfaces with this interface pattern**
- **Current recommended interface pattern for XI/PI specific scenarios via existing technical adapters (not using WS adapter)**
- **Only one operation allowed and name of service interface and operation must be identical**
- **XI runtime and XI adapter**

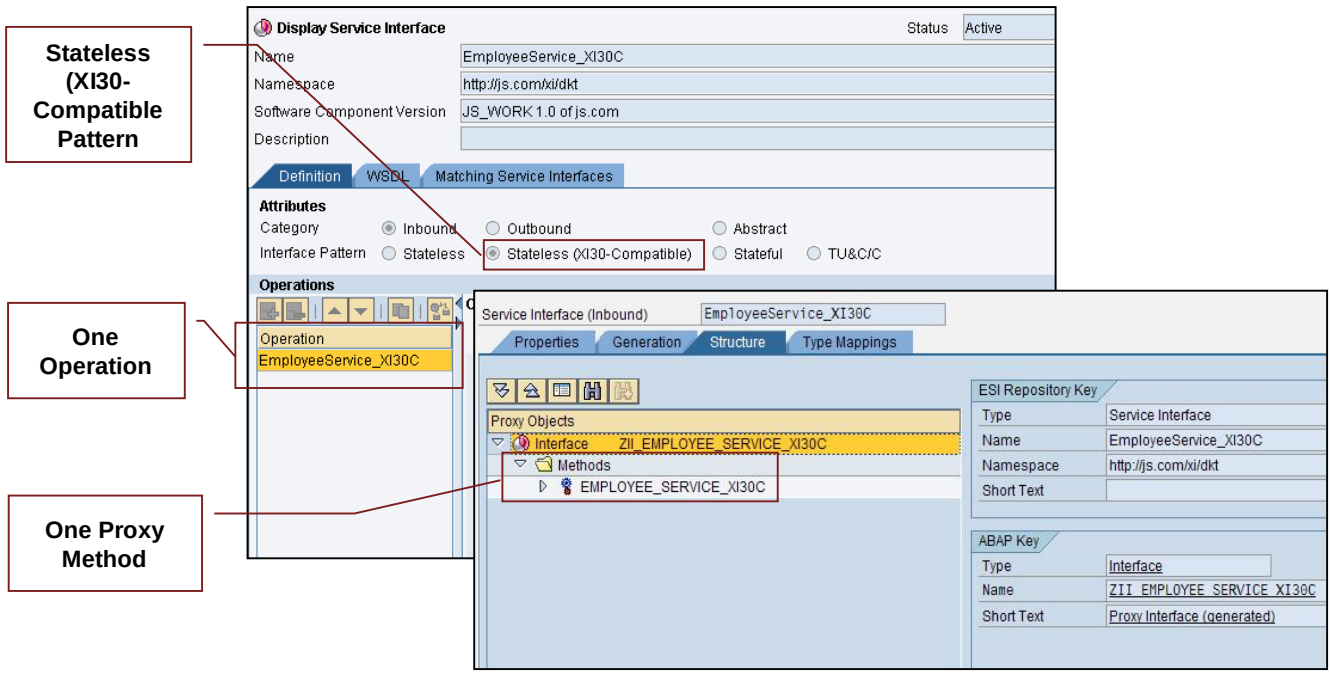
Additional Notes

- **For most enhanced/mediated scenarios, stateless or stateless (XI 3.0-compatible) pattern should be used**
- **Abstract interfaces can only use stateless or stateless (XI 3.0-compatible)**

- During service interface design, if it is known that an integration scenario is to be mediated via the Integration Server, the most likely interface patterns to use are *stateless* or *stateless (XI 3.0-compatible)*. These two interface patterns should account for the vast majority of integration scenarios that require mediated services within the integration server. The *stateless* interface pattern is also used in point-to-point (p2p) communication using web services.
- As listed here, the *stateless* interface pattern allows for single or multiple operations. It leverages the WS runtime and WS adapter in p2p and mediated communication.
- The *stateless (XI 3.0-compatible)* interface pattern is used for all existing NetWeaver 2004 and NetWeaver 2004s messages interfaces that are to be migrated to service interfaces. It is currently the recommended pattern for the XI/PI specific scenarios that use the common “technical adapters” such as file, jdbc, jms, etc. – basically all the adapters except the WS adapter. However, this pattern is limited to operation and the name of the service interface and operation must be identical. And of course, the XI runtime is used. The XI adapter is used certain scenarios as in the past – for example, when proxies are involved.
- As mentioned, most enhanced/mediated scenarios will use the stateless and stateless (XI 3.0-compatible) pattern. Also note that these are the only two interface patterns allowed for *abstract* interfaces.

Proxy Generation – Stateless (XI 3.0-compatible)

Proxy Generation Comparison



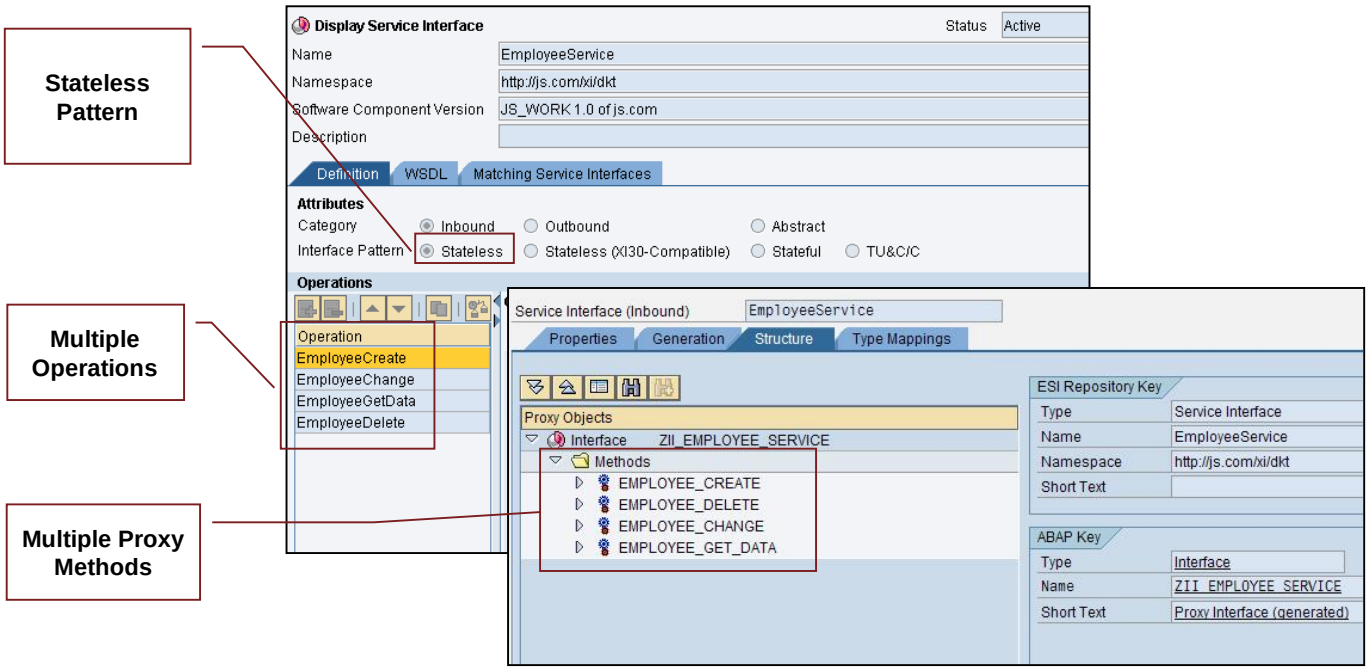
When multiple operations of a service interface are defined, proxy generation accounts for each of the operations as separate interface *methods* in the generated proxy definition.

Stateless (XI30-compatible) interface pattern with one operation:

- This pattern only allows for, at most, one operation. Plus, the operation name must be equal to the service interface name…
- When a proxy is generated, one corresponding proxy method is generated for the one operation within the service interface.

Proxy Generation – Stateless

Proxy Generation Comparison



When multiple operations of a service interface are defined, proxy generation accounts for each of the operations as separate interface *methods* in the generated proxy definition.

Stateless interface pattern with four operations:

- When the proxy is generated for this service interface, four separate proxy methods also get generated corresponding to each operation defined.
- No major surprise here, but mainly an illustration of what the proxy generation might look like for a service interface with multiple operations.

Interface Pattern – TU&C/C

Tentative Update & Confirm/Compensate (TU&C/C)

- **Reliable means to make synchronous update calls**
  - **Single or multiple update calls treated as one transaction**
- **Based on guaranteed delivery concept/mechanism for asynchronous messages**
- **Requires at least three messages:**
  - **one or more for the Tentative Update calls (synchronous)**
  - **exactly one Compensate (asynchronous)**
  - **exactly one Confirm (asynchronous)**
  - **additional Normal operations can be added (asynchronous or synchronous)**
- **Synchronous update calls are tentatively recorded until one of either Confirm (COMMIT) or Compensate (ROLLBACK) message is finally received.**
- **Web Service Runtime ensures delivery of Compensate message in all error or failure situations**
  - **compensate message registered before the first call of tentative update operation**
- **Both provider and consumer must understand protocol**
- **Only suitable in A2A scenarios, not B2B**

In terms of significance to PI, the interface patterns TU&C/C and, especially Stateful, are not very important. Nevertheless, it's helpful to at least touch on these interface patterns since these options are readily available in the UI.

TU&C/C stands for Tentative Update & Confirm/Compensate. This pattern has been developed to fill the gap in transactional behavior that currently exist today with synchronous messages. Synchronous messages, by their nature, cannot provide guaranteed delivery in and of themselves in cases of system or communication failure the way asynchronous messages can. Leveraging the guaranteed delivery capabilities of asynchronous messages, the TU&C/C pattern provides a reliable means to make one of more synchronous update calls in a transactional context.

At it's most basic level, the TU&C/C interface pattern requires three message:

- one for the *Tentative Update* calls (synchronous)
- exactly one *Compensate* (asynchronous)
- exactly one *Confirm* (asynchronous)

There can be more than one Tentative Update call. In addition, *Normal* operations can be added (asynchronous or synchronous).

The way it works is the following:

- One or more synchronous update calls are tentatively recorded until one of either a Confirm (COMMIT) or Compensate (ROLLBACK) message is finally received.
- Web Service Runtime ensures delivery of Compensate message in all error or failure situations
  - compensate message registered before the first call of tentative update operation
  - Both provider and consumer must understand protocol, only suitable in A2A scenarios, not B2B

Interface Pattern – Stateful

General

- **Successive calls use a state at the provider**
- **No guarantee of a common update of data at the receiver**
- **This interface pattern is only needed for a few special technical scenarios.**
- **This interface pattern cannot be used for enhanced/mediated communication using the Integration Server**
- **Only synchronous calls supported**

As mentioned, the *stateful* interface pattern is also not of much use within XI and consequently will not be discussed in detail. It will suffice to mention, in general, the following characteristics of this pattern:

- Successive calls use a state at the provider
- No guarantee of a common update of data at the receiver
- This interface pattern is only needed for a few special technical scenarios.
- This interface pattern cannot be used for enhanced/mediated communication using the Integration Server
- Only synchronous calls supported