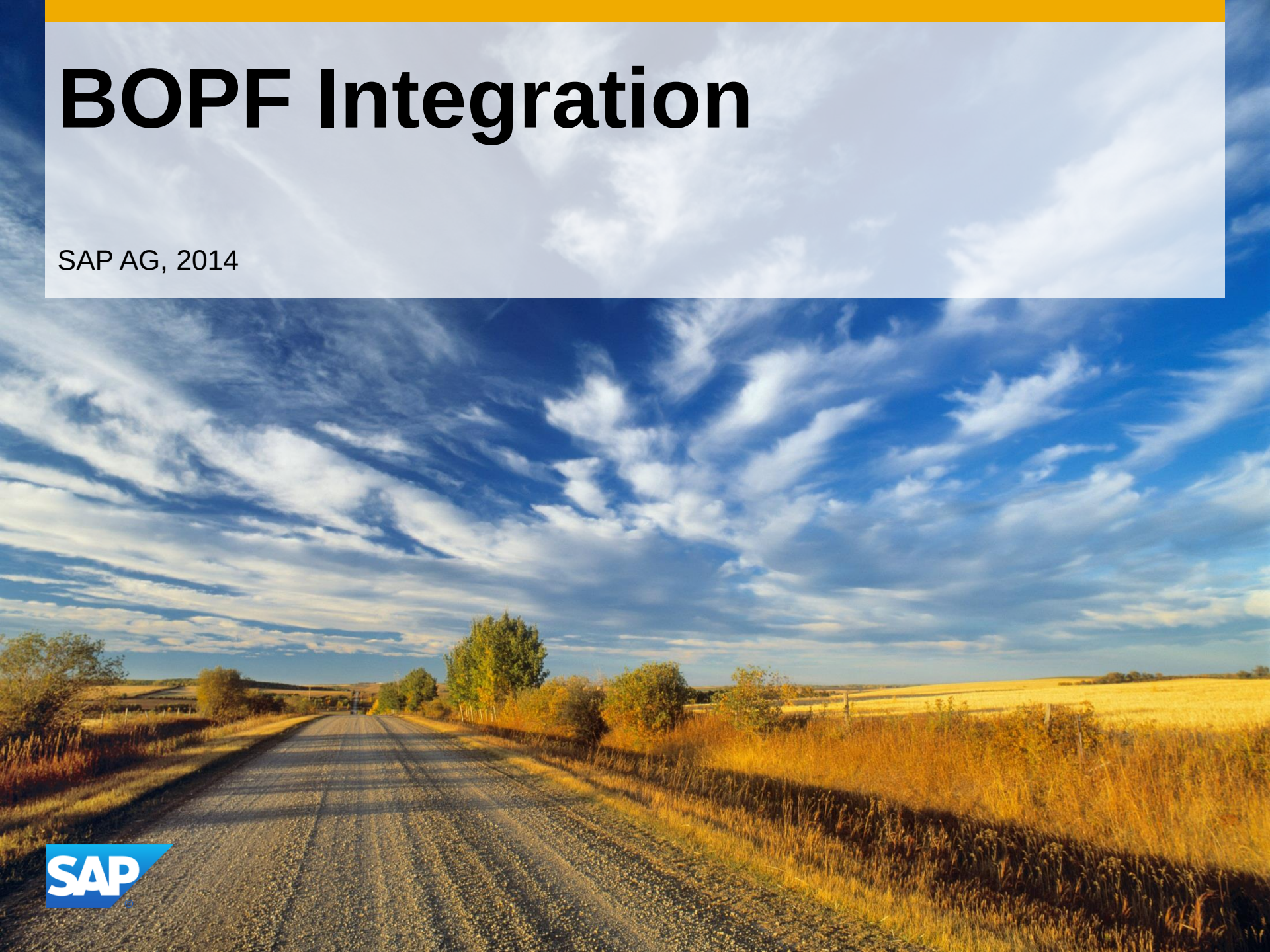
BOPF Integration



Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.

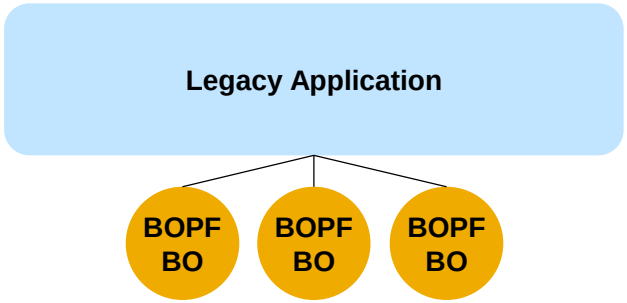
BOPF Integration
Agenda

1. **Integration of BOPF Business Objects into Legacy Applications**
   (Master/Slave Transaction Manager)
2. **Integration of Legacy Data and Functions into BOPF Business Objects**
   a. Function Integration on Application Layer
   b. Data Integration on Application Layer

BOPF Integration
Agenda

1. **Integration of BOPF Business Objects into Legacy Applications (Master/Slave Transaction Manager)**
2. Integration of Legacy Data and Functions into BOPF Business Objects
   a. Function Integration on Application Layer
   b. Data Integration on Application Layer (Legacy DAC)

Integration of BOPF Business Objects into existing Legacy Applications

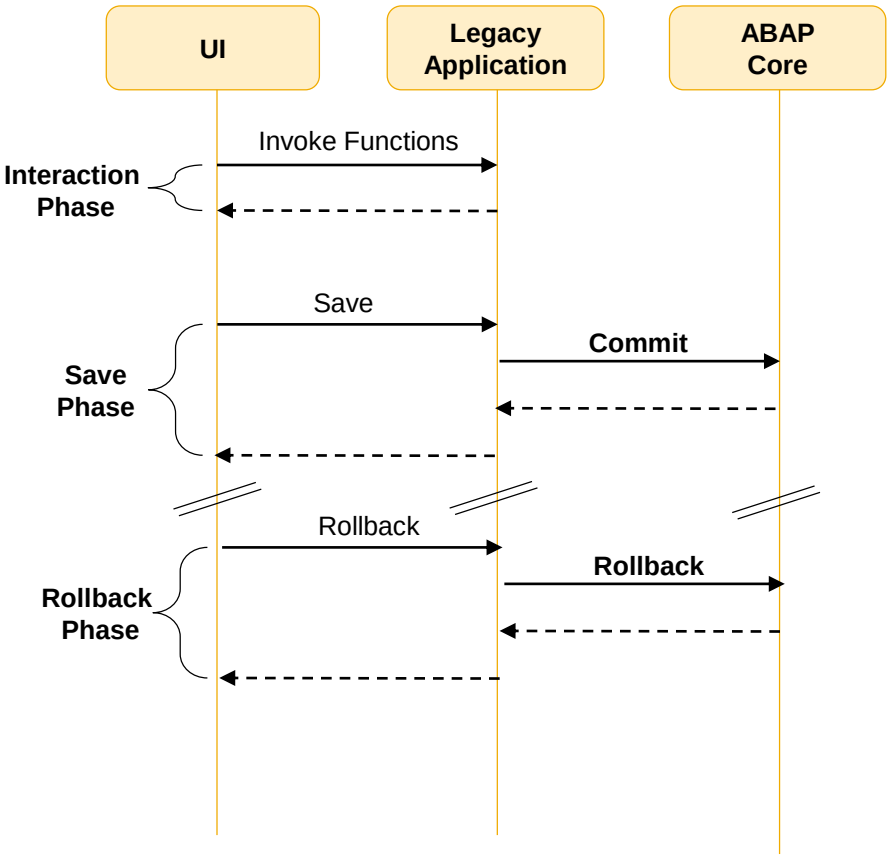


Scenario: Existing BOPF business objects needs to be integrated into an application or another framework.

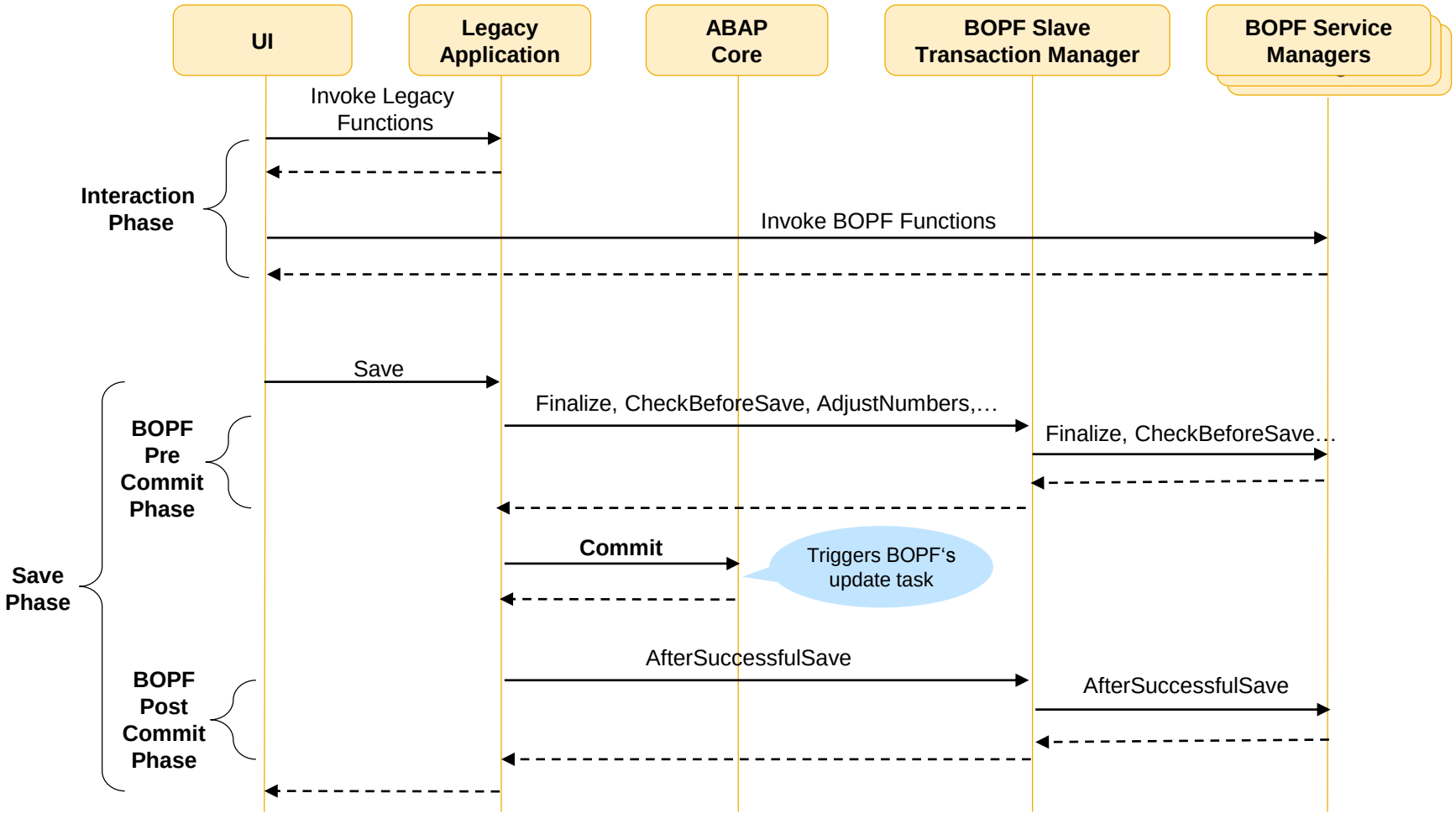
Issue: BOPF business objects must rely on the correct processing of the BOPF transaction phases like it is ensured using the BOPF standalone transaction manager. For instance the finalize phase must be executed to run the finalize determinations before a commit work is called.

Solution: Instead of using the (standalone) transaction manager, you have to implement an own „master“ transaction manager orchestrating both the save process of the legacy application as well as the save process of the BOPF business objects.

Integration of BOPF BOs into Legacy Applications (1/3)
Legacy Application

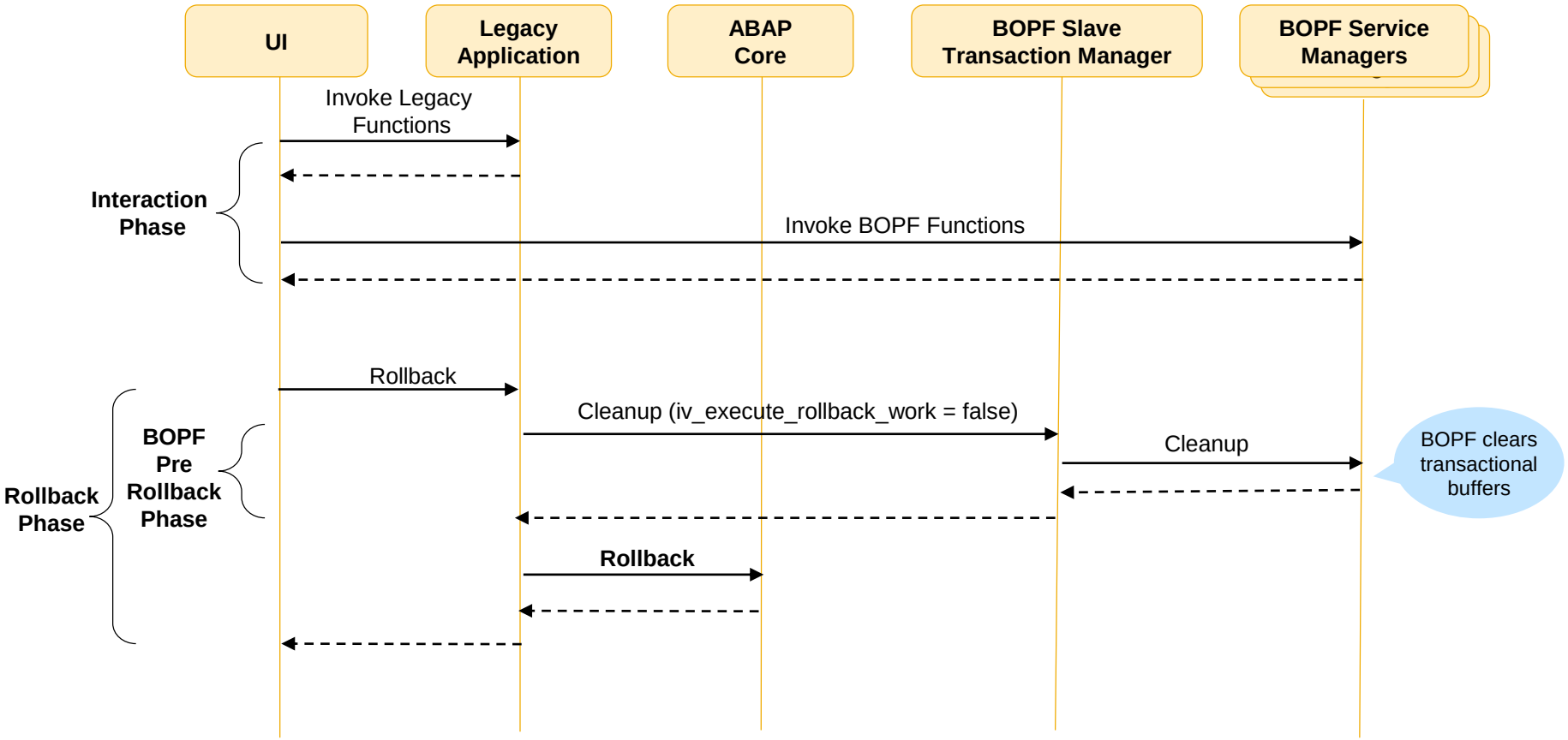


Integration of BOPF BOs into Legacy Applications (2/3)
Legacy Application & BOPF – Save Phase

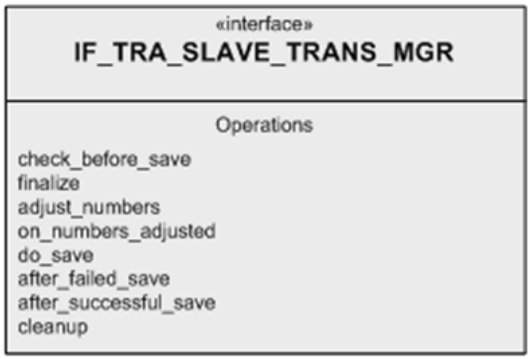


The Legacy Application is called Master Transaction Manager as it handles the COMMIT/ ROLLBACK.

Integration of BOPF BOs into Legacy Applications (3/3)
Legacy Application & BOPF – Cleanup Phase



Implementation of a Master Transaction Manager



Implement your own master transaction manager instead of using the BOPF (standalone) transaction manager. It needs to handle both the save (and cleanup) process of the BOPF BOs via the Slave Transaction Manager and the save of the legacy application. The slave manager instance can be received by the /BOBF/CL_TRA_TRANS_MGR_FACTORY => GET_SLAVE_TRANSACTION_MANAGER() method.

You can take the /BOBF/CL_TRA_TRANSACTION_MGR, method SAVE() as a reference of the sequence which needs to be called while save. Also implement the cleanup functionality.

BOPF Integration
Agenda

1. Integration of BOPF Business Objects into Legacy Applications (Master/Slave Transaction Manager)
2. **Integration of Legacy Data and Functions into BOPF Business Objects**
   a. Function Integration on Application Layer
   b. Data Integration on Application Layer (Legacy DAC)

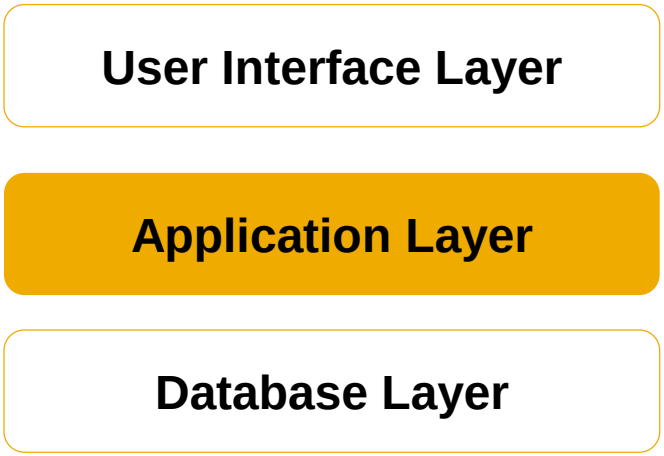
Integration Object

Data

Function

Depending on the scenario, (legacy) data and (legacy) functions needs to be integrated into a BOPF based application. For instance, an existing function module must be offered to the consumer by the help of a business object.

Integration Layer



Integration can be achieved on different layers of an application. In this chapter, only the application layer is focused as it is represented by the BOPF. The possibilities of integrating on User Interface Layer and Database Layer heavily depend on the used technology.

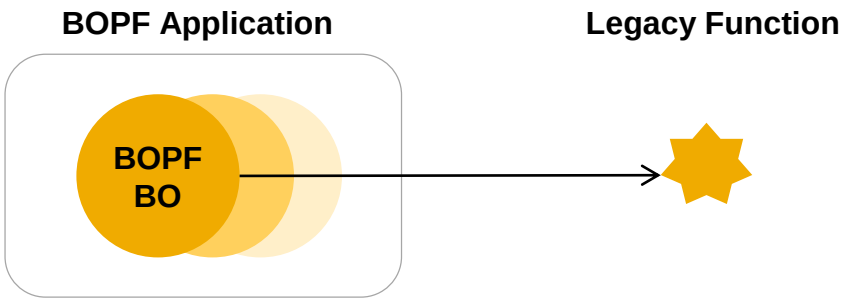
Integration Matrix
Examples of Integration Possibilities

	Data Integration (e.g. Database Table)	Function Integration (e.g. Function Module)
User Interface Layer	e.g. Frames in HTML	e.g. UI buttons calling functions
	Content of this Presentation	
Application Layer (BOPF)	Specific BOPF Data Access Class	Encapsulation via BOPF Actions, Determinations or Validations
Database Layer	e.g. DB Views	e.g. DB trigger, stored procedures

BOPF Integration
Agenda

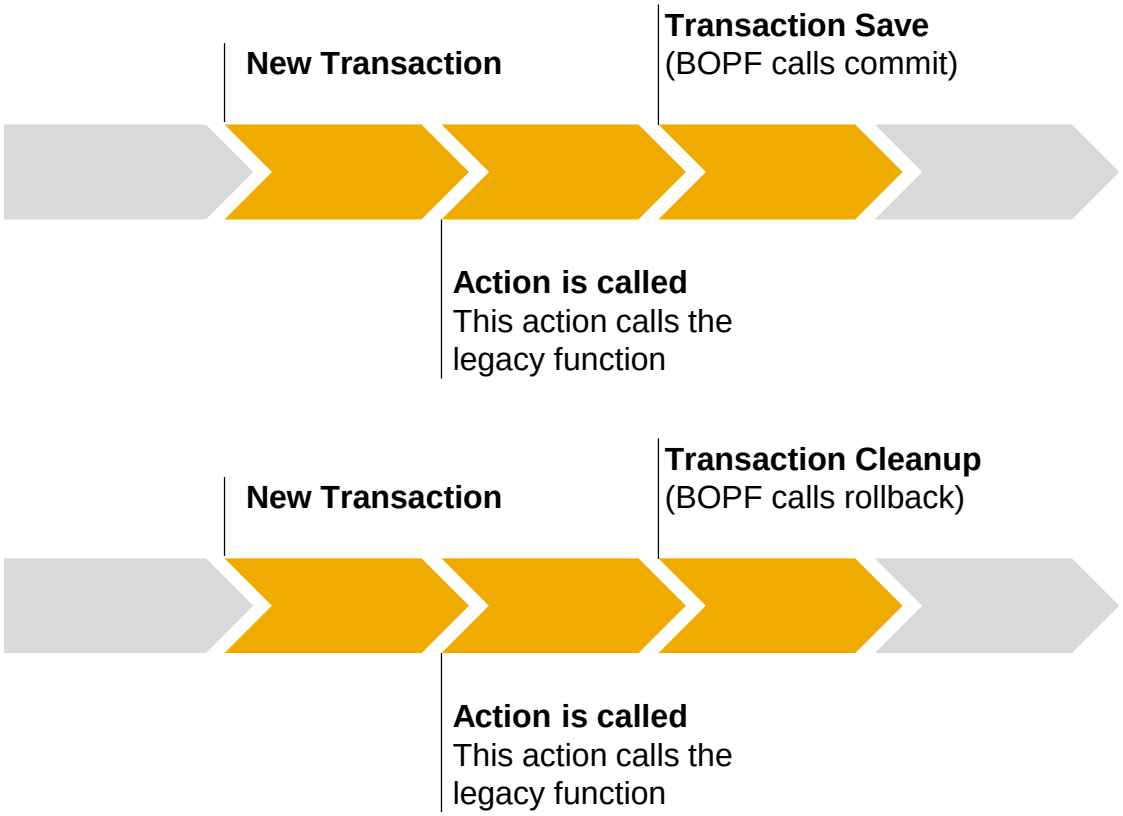
1. Integration of BOPF Business Objects into Legacy Applications (Master/Slave Transaction Manager)
2. Integration of Legacy Data and Functions into BOPF Business Objects
   a. **Function Integration on Application Layer**
   b. Data Integration on Application Layer (Legacy DAC)

Function Integration on Application Layer
Motivation



There is an existing function module which shall be consumed via a BOPF business object. This has the advantage that BOPF UI infrastructure can be reused (e.g. user interface frameworks) and the data can be easily consumed from other BOPF business objects.

Function Integration on Application Layer
Best Practise



The legacy function is encapsulated by the help of a BOPF action (determination or validation would also be possible). Changes done by that function are saved or cleaned up by the BOPF‘s commit or rollback.

Issues of Integrating Functions on Application Layers
Legacy Function calls Commit or Rollback

Issue
The function to be integrated uses transactional commits and rollbacks which can‘t be supressed by parameters.

Only the BOPF (or the master transaction manager if existing) is allowed to execute commits and rollbacks. If a function is calling those commands on ist own, the enqueues (update locks) set by BOPF to protect changed node instances are released. This can cause inconsistencies (lost updates) in case of a foreign session.

Solution: Staging
Stage the function calls in a static variable and execute the real function call only via an „after successful save“ determination. Use a second determination configured to the „cleanup“ time to clear that static variable if the current transaction is cleaned up by the user.

Issues of Integrating Functions on Application Layers
Legacy Function has no “cleanup”/”undo” Option

Issue
The function to be integrated does not have an undo option (transaction rollback would be not sufficient).

All changes which are done during a transaction must be made undone as soon as the user cleans up the current transaction instead of saving it. Also all changes done by that functions needs to be made undone if a rollback is not sufficient.

Solution 1: Staging
Stage the function calls in a static variable and execute the real function call only via an „after successful save“ determination. Use a second determination configured to the „cleanup“ time to clear that static variable if the current transaction is cleaned up by the user.

Solution 2: Log & Compensate
Create a determination at „cleanup“ time which compensates the changes done by the integrated function. Be aware that also the compensation operations could fail.

Issues of Integrating Functions on Application Layers
Legacy Function modifies Objects which needs to be locked

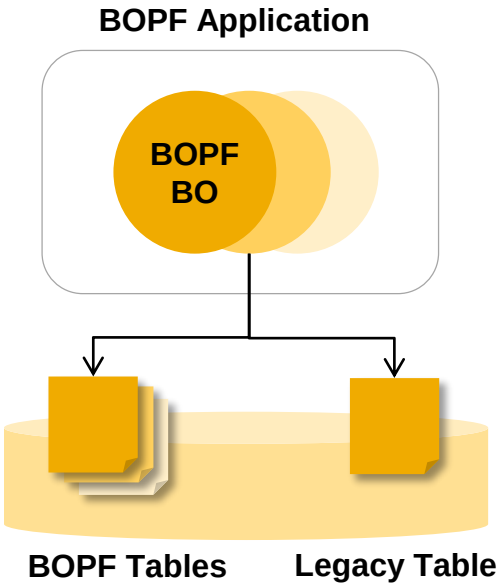
Issue
The function modifieds objects which needs to be locked and which might be locked by other users in parallel. If the function is called and the necessary lock is not available, the situation must not harm the transactional consistency (all or nothing paradigma)

Solution: Action Validation to get the Locks
If the function is integrated by the help of a BOPF action, you can use an action validation which acquires the necessary lock for your function up front. If the lock is not available, the action validation rejects the action execution.

BOPF Integration
Agenda

1. Integration of BOPF Business Objects into Legacy Applications (Master/Slave Transaction Manager)
2. Integration of Legacy Data and Functions into BOPF Business Objects
   a. Function Integration on Application Layer
   b. **Data Integration on Application Layer (Legacy DAC)**

Data Integration on Application Layer
Motivation



An existent legacy database table shall be integrated into a BOPF business object. It shall be represented by a node.

In contrast to directly access the table, the BOPF infrastructure (UI, extensibility) can be reused and further functionality can easily be added to the node.

Data Integration on Application Layer

Case 1: BOPF Default Persistency Layout is used

Root Database Table

Field	Type
MANDT	MANDT
DB_KEY	/BOBF/CONF_KEY (RAW16)

…

Subnode Database Table

Field	Type
MANDT	MANDT
DB_KEY	/BOBF/CONF_KEY (RAW16)
PARENT_KEY	/BOBF/CONF_KEY (RAW16)

…

Subsubnode Database Table

Field	Type
MANDT	MANDT
DB_KEY	/BOBF/CONF_KEY (RAW16)
PARENT_KEY	/BOBF/CONF_KEY (RAW16)
ROOT_KEY	/BOBF/CONF_KEY (RAW16)

…

If the legacy database table has already compliant fields to the BOPF default persistency layout (very rare case) or those fields can be introduced, the integration is possibly by just maintaining the corresponding table in the node configuration.

Data Integration on Application Layer Issues
Case 1: BOPF Default Persistency Layout is used

Issue

The legacy database table is usually used by another application. This application uses an own enqueue concept which needs to be obeyed. Before the BOPF modifies the entries of that legacy database table, a corresponding enqueue must be acquired (and later on released).

Solution

Reimplement the BOPF Lock and Unlock Action for the node that integrates the legacy table. Set both the locks for the legacy application and the ones for the BOPF. As soon as BOPF requires locks, your lock/unlock action will be invoked to acquire or release the lock.

Data Integration on Application Layer Issues
Case 2: Incompatible Database Layout

Issue
The primary key of the legacy database table is type-compliant to BOPF default persistency layout. However the name does not match the BOPF default persistency layout (e.g. the key not named „DB_KEY“). It is not possible to change the field‘s name.

Solution 1: Persistency Mapping (not recommended)
Use the persistency mapping tab of the BOPF Configuration UI to bind the different named field to the DB_KEY field. This is not possible in BOBX and Eclipse and should not be done at all.

Solution 2: Own DAC (not recommended)
Implement your own DAC by copying /BOBF/CL_DAC_TABLE and replace the SQL statements/coding accordingly. As this dublicates a lot of code which needs to be separatly maintained, this should not be done.

Solution 3: Usage of Legacy DAC
The Legacy DAC allows to integrate legacy database tables having arbitrary fields as a common BOPF node.

Legacy DAC
How to use

1. Create a persistent node and maintain the BOPF Legacy DAC class as data access class
2. Maintain a unqiue alternative key namend „DB_KEY“ on that node consisting of all the components of the legacy database table building the legacy key (can be more than one component)
3. If the node is not the ROOT node, maintain also a unique alternative key „PARENT_KEY“ (consisting of all legacy components building the parent key information) and the corresponding „ROOT_KEY“. If there is no ROOT_KEY alternative key, the legacy DAC will automatically concatenate the parent key information to simulate a ROOT navigation.
4. In case of self implemented queries, the mapping functionality of the legacy DAC can be usedvia ist methods map_to_bopf_key and map_to_legacy_key.

Legacy DAC
Restrictions

- Please be aware that in this case an own lock and unlock action class must be provided as the BOPF library lock class expects globally unique keys – those transient keys are only unqiue within their session but not cross session (see chapter „Action“ for action implementations). But as there is usually non-BOPF business logic operating on the legacy database table, the lock/unlock behavior always needs to be adapted.
- Please be aware that the transient keys must not be stored by the consumer as they will change for each session. Thus the consumer always has to start the transaction using an alternative key (or query) to get the current transient keys but is not allowed persist the transient keys.

Legacy DAC
Principle

Consumer

KEY	ID	ID_TYPE	CREATED_BY
0	23	INV	SMITH
1	24	CDV	CARTER

KEY field is hidden by field properties (disabled)

BOPF

KEY	ID	ID_TYPE	CREATED_BY
0	23	INV	SMITH
1	24	CDV	CARTER

Legacy DAC
+Mapping Table

KEY	ID	ID_TYPE
0	23	INV
1	24	CDV

KEY	ID	ID_TYPE	CREATED_BY
0	23	INV	SMITH
1	24	CDV	CARTER

Legacy Database Table

ID	ID_TYPE	CREATED_BY
23	INV	SMITH
24	CDV	CARTER

Field “ID” and “ID_TYPE” are the primary key components of the legacy table uniquely identifying each table entry

Thank you

© 2012 SAP AG. All rights reserved.