Service Provision and Exposure

Applies to

SAP Banking Solutions with Banking Services 6.0 and NetWeaver 7.0. For more information, visit the Banking homepage or the Web Services homepage. .

Summary

This document has been created to provide guidance to IT Architects, Solution designers and Business Process Experts the context and background to integrating SAP Banking Services based on the NetWeaver platform.

This is not intended to be an exhaustive solution outline, but instead it is intended to provide a starting point to your discussions developing the integration architecture of Banking Services into the heterogeneous environment of a bank covering the main aspects of data types, existing and newly defined interfaces and the implementation requirements that will alter your adoption of each interface type beyond a general discussion on semantics in Service Oriented Architecture.

The main integration mechanisms related to Banking Services will be described in turn, with 'rules of thumb' provided to accelerate project decisions.

Authors: Dr.Thomas Schindler, Gavin Targonski

Company: SAP

Created on: 6 December 2008

Author Bio

Dr. Thomas Schindler is the chief solution architect for consulting in the banking line of business. Thomas' main responsibility is guiding and leading architecture definition within customer environments for the Global Banking Line of Business. He is also a contributor to the Banking Industry Architecture Network (BIAN) defining SOA in Banking. Thomas joined SAP in 1997 and has held various positions in consulting and now within the SOA Engineering team. Thomas can be reached at: Thomas.schindler@sap.com

Gavin Targonski works as a technical architect in the SOA Engineering Group within SAP's global line of business banking. His main responsibility is designing architectures which allow customers to link SAP's Enterprise SOA-based SAP for Banking solutions to a bank's application landscape. He joined SAP in 2006 and has held various responsibilities in the area of SAP Consulting. Gavin can be reached at Gavin.Targonski@sap.com

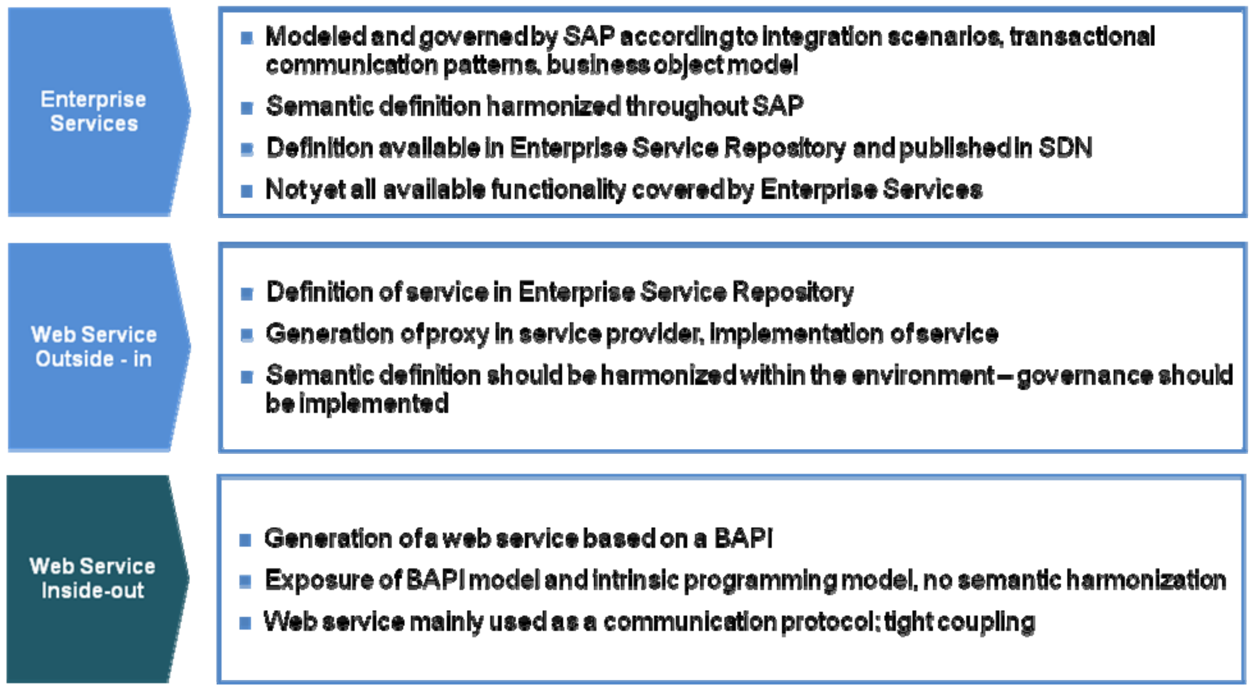
Table of Contents

Introduction

This document has been created to provide guidance to IT Architects, Solution designers and Business Process Experts and to provide a summary of the main integration mechanisms related to the Banking Services solution from SAP. The main interface mechanisms will be described in turn, with ‘rules of thumb’ provided to accelerate project decisions.

This is not intended to be an exhaustive solution guide, but as a starting point to your discussions when implementing the SAP Banking platform into an existing IT landscape.

The main integration approaches in this document are summarised here:



Enterprise Services	▪ Modeled and governed by SAP according to integration scenarios, transactional communication patterns, business object model ▪ Semantic definition harmonized throughout SAP ▪ Definition available in Enterprise Service Repository and published in SDN ▪ Not yet all available functionality covered by Enterprise Services
Web Service Outside - in	▪ Definition of service in Enterprise Service Repository ▪ Generation of proxy in service provider, implementation of service ▪ Semantic definition should be harmonized within the environment – governance should be implemented
Web Service Inside-out	▪ Generation of a web service based on a BAPI ▪ Exposure of BAPI model and intrinsic programming model, no semantic harmonization ▪ Web service mainly used as a communication protocol; tight coupling

This document provides an overview and recommendations to interfacing approaches for Banking Services from SAP (currently release 6.0), and SAP business applications in general, regarding semantics. The topic areas covered fall into three categories:

- Integration type comparison
- Data type comparison
- Options for adoption

This document describes the various SAP service consumption options while combining the communication protocol and data models into options for use within an implementation project context. The focus is on extending given functionality of SAP standard by project specific implementation approaches. The body of this document will defocus on the main implementation decisions, together with discussion and guidance on data types and interface implementation. We will specifically not be focussing on the general discussion on semantics in SOA and on the model driven approach for service definition, although we will outline the best practice for interface implementation in a project environment in section 7.

Reference information is added in appendices.

This document does not cover

- Service modelling
- Documentation on BAPI and Enterprise Services
- Technical features of Enterprise Service (i.e. idempotency)

- Web Services Security
- Connectivity
- Performance recommendations

This document is focussed on supporting architecture decisions and is written from an implementation project view. Therefore the diction is related to terms used in typical implementation projects.

Definitions

As an introduction to the document it is first useful to clarify what is understood by the term ‘Service’. A service is a resource which performs a consistent set of tasks based on a consumer request, thus done via standardized interfaces.

The approaches of SAP providing functionality to consumers and their access are described in this section. However, for a more comprehensive definition we refer to the documentation[1].

Web Service (WS)

A web service is a service which is accessible via a web interface.

A definition of web services can be found by W3C:

‘Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.’ [2]

Web services use SOAP as communication protocol.

Enterprise Service (ES)

Enterprise Services are defined by SAP as “web services which include a high level of functionality and which support business processes”. The SAP defined Enterprise Services are modelled with Business Objects and Process Components and use transactional communication patterns. Definition of SAP Enterprise Services is via a governance process called PIC (Process Integration Content) to allow for semantic harmonization throughout SAP and adoption of standards. Their semantic definition is stored in the Enterprise Service Repository.

The Enterprise Services Repository is available in SAP Process Integration and SAP Composition Environment. To access Enterprise Services in a mediated way the Enterprise Services Registry based on UDDI 3.0 is located in SAP Process Integration.

This is in contrast to BAPI’s, Enterprise Services are specifically intended to support loose coupling of provider and consumer by delivering transactional integrity and idempotency by design.

[1] http://help.sap.com/saphelp_banking60/helpdata/en/f5/416c4255b3c553e10000000a1550b0/frameset.htm

and www.sdn.sap.com

[2] http://www.w3.org/

Web Service versus Enterprise Service

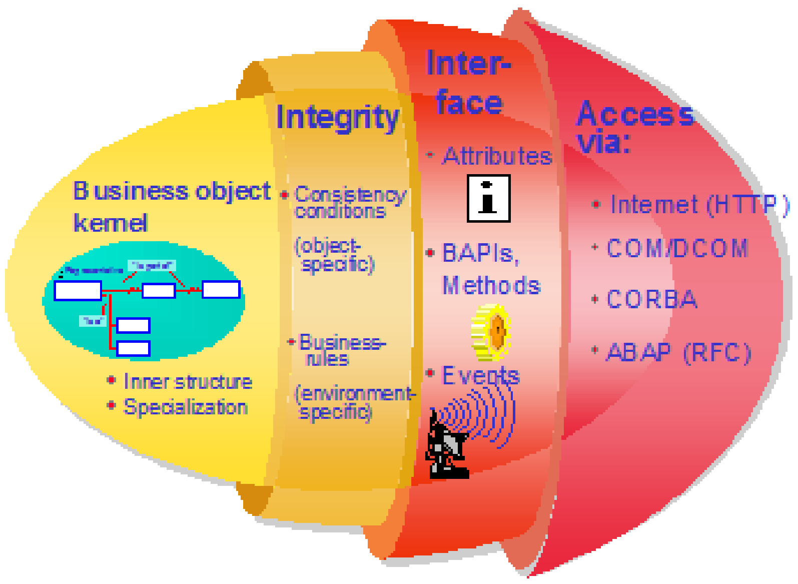
The main differences between a Web Service and an Enterprise Service from a data type perspective are:

1. Top-Down vs. Bottom-Up approach: web services outside-in are modelled from a business process perspective down to the level of the interface operation and its attributes, while web services inside-out can be just the technical means of exposing an existing ABAP functionality in a standard non-proprietary manner.
2. Enterprise Services are modelled in the ESR and use Global Data Types (GDT's), which are a uniform and consistent typing across SAP's Enterprise Services and represent a semantic business vocabulary. The GDTs can be extended by user-defined types. Web services inside-out, on the other hand, use the basic W3C data types with out any inherent business semantics, thus providing the internal data model to the consumer.

BAPI

The BAPI (Business Application Programming Interface) is a well defined interface and underpins the programming rules and framework of SAP. BAPIs provided by SAP standard software are meant to be stable through release upgrades.

BAPI's are based on a business objects model, which is defined in the BOR (Business Object Repository). The BAPIs describe which functions (services, operations) the business objects make available to the consumers and are 'methods' on the business objects.



While there is a broad range of functionality provided via the 'traditional' BAPI interface on the current Banking Services platform (6.0), the requirements for open connectivity, transactional integrity and idempotency are not easily delivered using this interface mechanism. New functionality will therefore be delivered through Enterprise Service definitions, leveraging the SAP NetWeaver platform to deliver the standards based interoperability in heterogeneous IT landscapes. Enterprise Services expose the SAP Banking 'objects' through well defined interfaces and web services standards.

Wrapping Existing Functionality

Process Object

Existing ABAP functionality can be leveraged in a well composed and architected '*process object*' layer. This newly defined (out-side in) *process object* is delivered on top of the pre-defined & pre-delivered ABAP/BAPI interfaces to allow:

- Object oriented service consumption.
- Clustering of functionality to business domains.
- Enable functionality aggregation (a composition of several existing functions) to the correct granularity.
- Hide the complexity of BAPI stateful nature with the need to commit in a different call. Everything will be done in this layer to make it stateless (and web service ready).

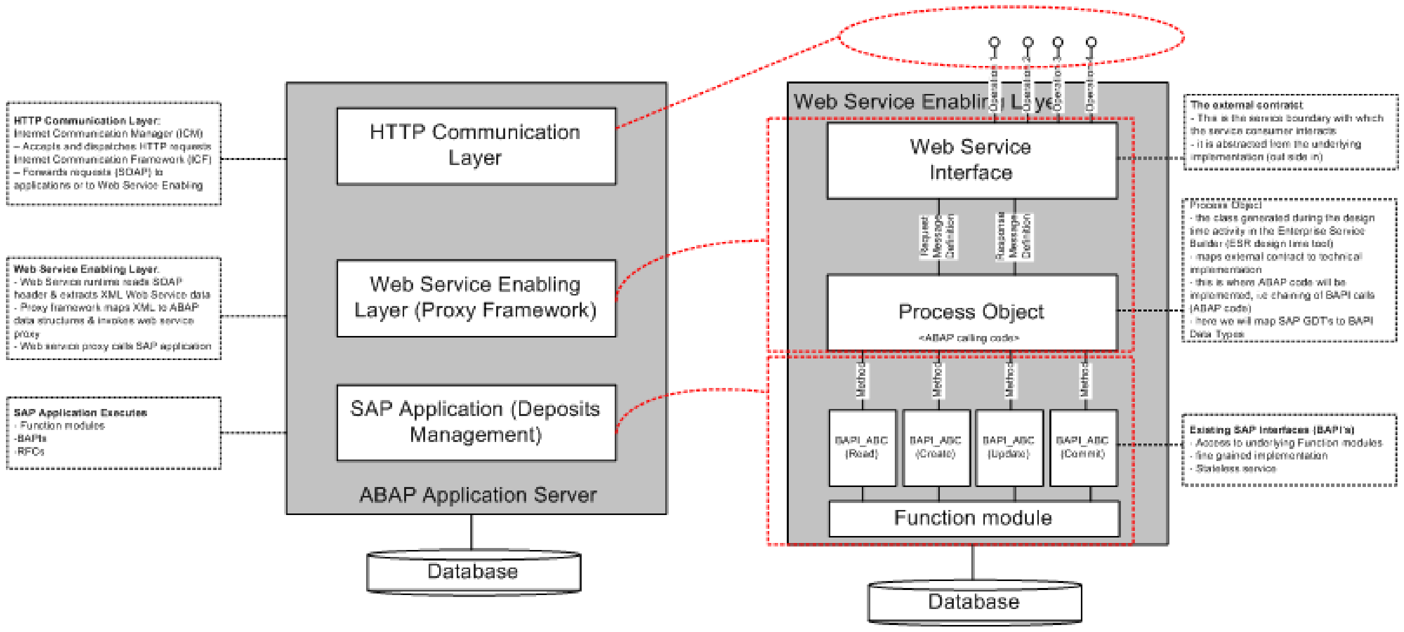
This layer includes Object Oriented classes called, for the purpose of this document, *process objects*. Usually these objects make use of several BAPI calls and are implemented as ABAP classes.

SAP recommends to use Enterprise Services whenever they are available. Enterprise Services are modelled according to the SAP Banking Services underlying service model and are governed and implemented by the SAP rules and guidelines. As with BAPI's enterprise services are kept stable through release upgrades.

The implementation of the interfaces in turn relies upon the data types that exist within the SAP banking platform. Using data types provides two key benefits:

- Defined structure & definitions to mapping customer data model to SAP data model
- Guaranteed stability: changes can me mapped and therefore managed
- Re-usable process objects and re-usable interfaces assured via the model driven approach

The data types from SAP are described below along with examples of each.


HTTP Communication Layer:
Internet Communication Manager (ICM)
– Accepts and dispatches HTTP requests
Internet Communication Framework (ICF)
– Forwards requests (SOAP) to applications or to Web Service Enabling
Web Service Enabling Layer:
- Web Service runtime reads SOAP header & extracts XML Web Service data
- Proxy framework maps XML to ABAP data structures & invokes web service proxy
- Web service proxy calls SAP application
SAP Application Executes
- Function modules
-BAPIs
-RFCs
HTTP Communication Layer
Web Service Enabling Layer (Proxy Framework)
SAP Application (Deposits Management)
ABAP Application Server
Database
Operation 1
Operation 2
Operation 3
Operation 4
Web Service Enabling Layer
Web Service Interface
Request Message Definition
Response Message Definition
Process Object
<ABAP calling code>
Method
Method
Method
Method
BAPI_ABC (Read)
BAPI_ABC (Create)
BAPI_ABC (Update)
BAPI_ABC (Commit)
Function module
Database
The external contract:
- This is the service boundary with which the service consumer interacts
- it is abstracted from the underlying implementation (out side in)
Process Object
- the class generated during the design time activity in the Enterprise Service Builder (ESR design time tool)
- maps external contract to technical implementation
- this is where ABAP code will be implemented, i.e chaining of BAPI calls (ABAP code)
- here we will map SAP GDT's to BAPI Data Types
Existing SAP Interfaces (BAPI's)
- Access to underlying Function modules
- fine grained implementation
- Stateless service

Data Types

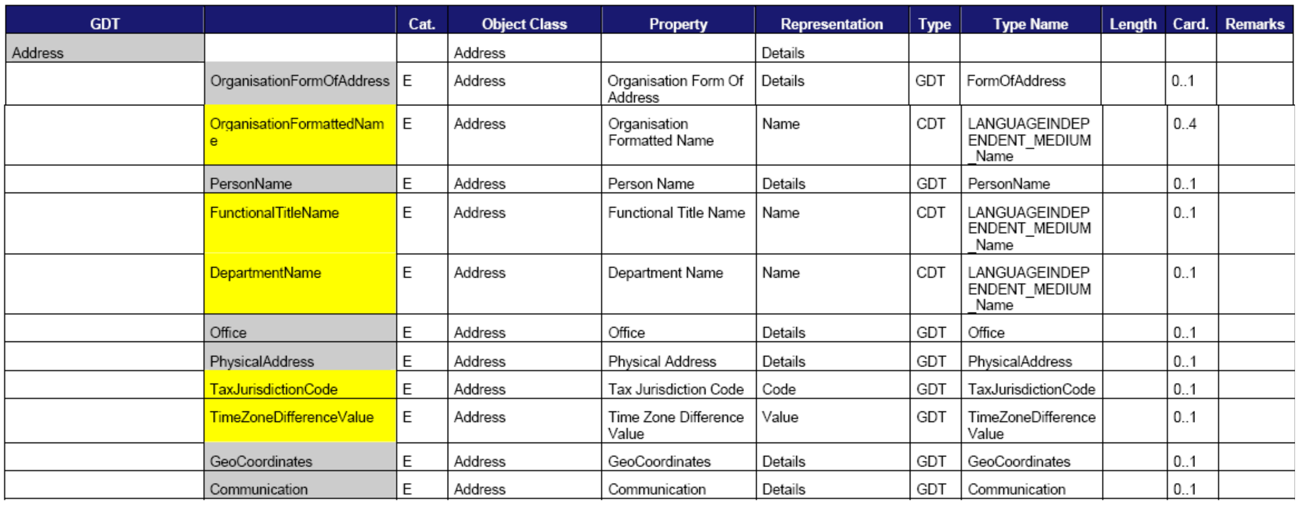
SAP’s Global Data Types

To achieve a high level of reusability for data types across it applications, SAP uses global data types (GDTs). GDTs are SAP-wide normalized and reconciled data types with business-related content and are based on standards. GDTs ensure harmonization of the hierarchical signature structures of operations and the structure of the business object nodes. They represent a business-related subject matter that is described by a specified structure. Global data types are based on the web services description provided by WSDL.

A recommendation is to use SAP’s Global Data Types for web services whenever they exist to satisfy the requirements to ensure compliance to SAP and minimize implementation effort. Otherwise, a new user-defined data types will be created.

Customer data types and structures will be mapped to SAP data types during the design time work. The SAP web application server will then be the runtime mechanism through which the ABAP Class invoked by this service transforms the GDTs to the ABAP data types prior to calling the internal BAPIs or ABAP classes.

Example[3]:



GDT		Cat.	Object Class	Property	Representation	Type	Type Name	Length	Card.	Remarks
Address			Address		Details					
	OrganisationFormOfAddress	E	Address	Organisation Form Of Address	Details	GDT	FormOfAddress		0..1	
	OrganisationFormattedName	E	Address	Organisation Formatted Name	Name	CDT	LANGUAGEINDEPENDENT_MEDIUM_Name		0..4	
	PersonName	E	Address	Person Name	Details	GDT	PersonName		0..1	
	FunctionalTitleName	E	Address	Functional Title Name	Name	CDT	LANGUAGEINDEPENDENT_MEDIUM_Name		0..1	
	DepartmentName	E	Address	Department Name	Name	CDT	LANGUAGEINDEPENDENT_MEDIUM_Name		0..1	
	Office	E	Address	Office	Details	GDT	Office		0..1	
	PhysicalAddress	E	Address	Physical Address	Details	GDT	PhysicalAddress		0..1	
	TaxJurisdictionCode	E	Address	Tax Jurisdiction Code	Code	GDT	TaxJurisdictionCode		0..1	
	TimeZoneDifferenceValue	E	Address	Time Zone Difference Value	Value	GDT	TimeZoneDifference Value		0..1	
	GeoCoordinates	E	Address	GeoCoordinates	Details	GDT	GeoCoordinates		0..1	
	Communication	E	Address	Communication	Details	GDT	Communication		0..1	

External data types

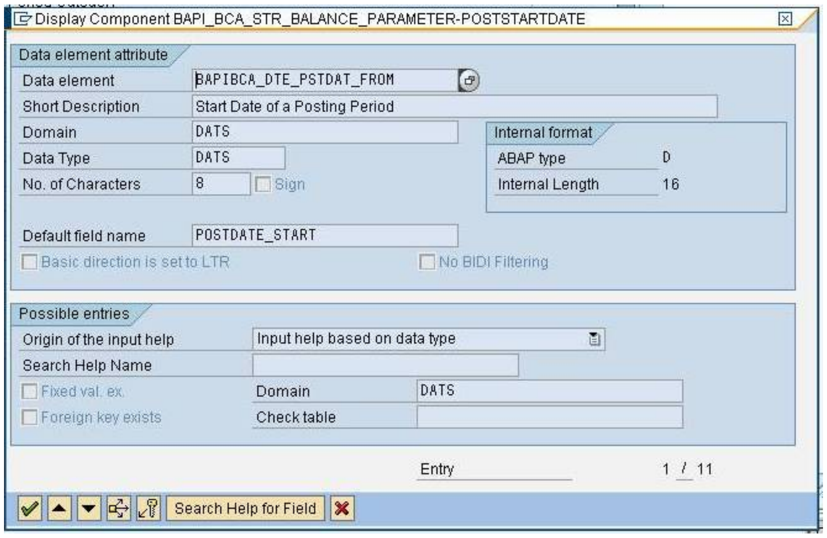
External data types as provided by an alternative vendor or framework can either be modelled into user-defined data types to be used with a Web Service, or used in an interface of an ABAP Class.

ABAP dictionary data types

These are the basic data types used within ABAP classes and are also used by BAPIs.

[3] Data Type Catalog - Definitions of Global Data Types and Core Data Types p615 (of 7952)

Example[4]:


Display Component BAPI_BCA_STR_BALANCE_PARAMETER-POSTSTARTDATE

Data element attribute

Data element	BAPIBCA_DTE_PSTDAT_FROM
Short Description	Start Date of a Posting Period
Domain	DATS
Data Type	DATS
No. of Characters	8
Default field name	POSTDATE_START

Sign

Basic direction is set to LTR

No BIDI Filtering

Internal format

ABAP type	D
Internal Length	16

Possible entries

Origin of the input help	Input help based on data type
Search Help Name	
Domain	DATS
Check table	

Fixed val. ex.

Foreign key exists

Entry 1 / 11

These data types are based on the programming model of the business functionality within the platform and are therefore SAP proprietary.

[4] BAPI explorer – SAP Banking Services

Communication Protocol Comparison

RFC

Pros:

- Best Performance since it does not have the overhead of the SOAP processing. Communication overhead is minimised using RFC over Binary, and can be implemented using various connectors (e.g. from 3rd party integration software providers)
- Good old proven practice since using SAP's services until web services were introduced. Many implementations in the market place.

Cons:

- Non-standard and proprietary protocol, which is not supported by all vendors.
- Not aligned with SAP's vision and long term roadmap of using Enterprise Services. SAP is exposing increasing amounts business functionality via Enterprise Services. This means that for applications that are web service enabled, there will be migration effort when using RFC interfaces, and additional modelling work required to migrate to the new interfaces.
- No seamless version management for functions – management based on SAP transports and testing tools.
- Service consumer has to manage the changes to RFC directly as there is no abstraction from the underlying implementation. Abstraction is provided by the customer web service implementation or the Enterprise Service implementation from SAP.
- Connection pooling is not possible within the SAP platform. Reliance must be made on connection handling in the middleware layer.
- Security: web service security standards are implemented via the SOAP runtime on the underlying NetWeaver application server. These standards are not however supported through the BAPI/RFC based interfaces. For the detailed security support please see the following reference: NetWeaver Security - SAML[5]

Web Services inside-out

Pros:

- Version management possible when using the ESR (Service Builder) for creating the web service proxy
- Faster to implement than Web Services outside-in in technical terms – shortest way to expose functionality as a web service without the need to model Process Components and Business Objects.
- Standard binding via UDDI repository

Cons:

- Service consumer has to manage the changes to the web service inside-out directly as there is no abstraction from the underlying implementation, when the web service has been generated from the web application service by using the development workbench.
- If, in the future, the project decides to use Enterprise Services or web services outside-in, choosing this option will incur in significant modelling effort and will involve code changes (exposing functionality in a different way due to the top-down modelling approach of Enterprise Services).

[5] http://help.sap.com/erp2005_ehp_03/helpdata/EN/3e/69d8dcd8d08f4d8f864be86404056f/frameset.htm

- Closely coupled to the underlying technical implementation which means service consumers will have to manage underlying changes. This will drive the need for a middleware (service de-coupling) layer.

Web Service Outside-In

Pros:

- Business (service-consumer) driven abstraction from technical implementation. Enterprise Services Repository provides tooling support to enforce design time governance of Enterprise Services.
- Model driven design is beneficial as the service consumers are abstracted from the provider service definitions. If modelling takes into account the SAP service model it enhances stability at release upgrades.
- Version management of Enterprise Services is supported through standard SAP tooling ESR.
- WS standard implementation (discovery / binding) via UDDI repository.

Cons:

- A significant effort of Top-Down modelling is required even when using SAP's model as a starting point due to the fact that it is not complete. However models will evolve with time.
- Performance of SOAP handling and XML marshalling is significantly slower than the native protocol like RFC.
- Requires modelling overhead to create the service.

Data Types Comparison

ABAP Dictionary data types

Pros:

- No transformation needed from the externally exposed interface 'data object types' to the internal ABAP data types functions, and vice versa.
- Reference implementations & consulting experience is commodity; SAP functional consultants are most knowledgeable with these 'traditional' data types.

Cons:

- Most complex in mapping – based on inside out (technical implementation) and focussed on the internal programming model.
- These are mainly technical data types with business semantics attached, resulting in possible inconsistency between existing BAPI interfaces in terms of data typing – same subject matter might be described by different data types. Banking Services has been designed to avoid this issue.
- Several interfaces receiving (or returning) the same information could do so with different signatures.
- To ensure consistency, a 'robust' governance & release mechanism should be implemented for designers and developers to work within.

Global Data Types

Pros:

- Usage of GDTs (or for this matter any data type model) brings consistency to the service interfaces.
- SAP's strategy for service modelling will bring focus on introducing all data types as Global Data Types thus encouraging external consultants and integration providers to understand and work with Global Data Types as compared to the internal ABAP dictionary data types.
- For the existing banking Global Data Types used within existing SAP Enterprise Services there is a conversion (runtime) routine (provided within an ABAP class) from the GDT to the ABAP dictionary data types. This ensures consistency of interaction without having to create custom code.
- Global Data Types are defined through an internal SAP governance process (SAP PIC process), which takes care that industry standards are used for definition.

Cons:

- Not yet a complete data type model available. Due to the fact that only a portion of the banking functionality is released as Enterprise Services new user-defined Data Types would need to be implemented. A governance process would need to be implemented within the project to produce these extended data types based on the catalogue already available[6].
- GDT model not wholly industry specific, but it is cross industry defined. SAP works together with other banks and providers to define an industry-side alignment on semantics in banking (BIAN). The Data types used within the banking platform will be aligned to this industry defined semantic model.

[6] https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/303fd192-1db2-2a10-59b9-9dfd93f4b10f

SAP Functionality Consumption Options

Almost all permutations of the communication protocols and data types are possible. Following are the available options regarding provisioning of SAP functionality:

- Enterprise Services with Global Data Types: harmonise and adopt SAP definitions
- Web services outside-in with Global Data Types: import customer defined data types or extend GDT/CDT/ABAP dictionary data types.
- Web Services inside-out with customer defined data types or ABAP dictionary data types: expose interfaces's already available on the platform and map to the interface.
- RFC with customer defined data types or ABAP dictionary data types: harmonise and map to existing data types.

Following options will not be explored further in this document as they are not considered strategically viable:

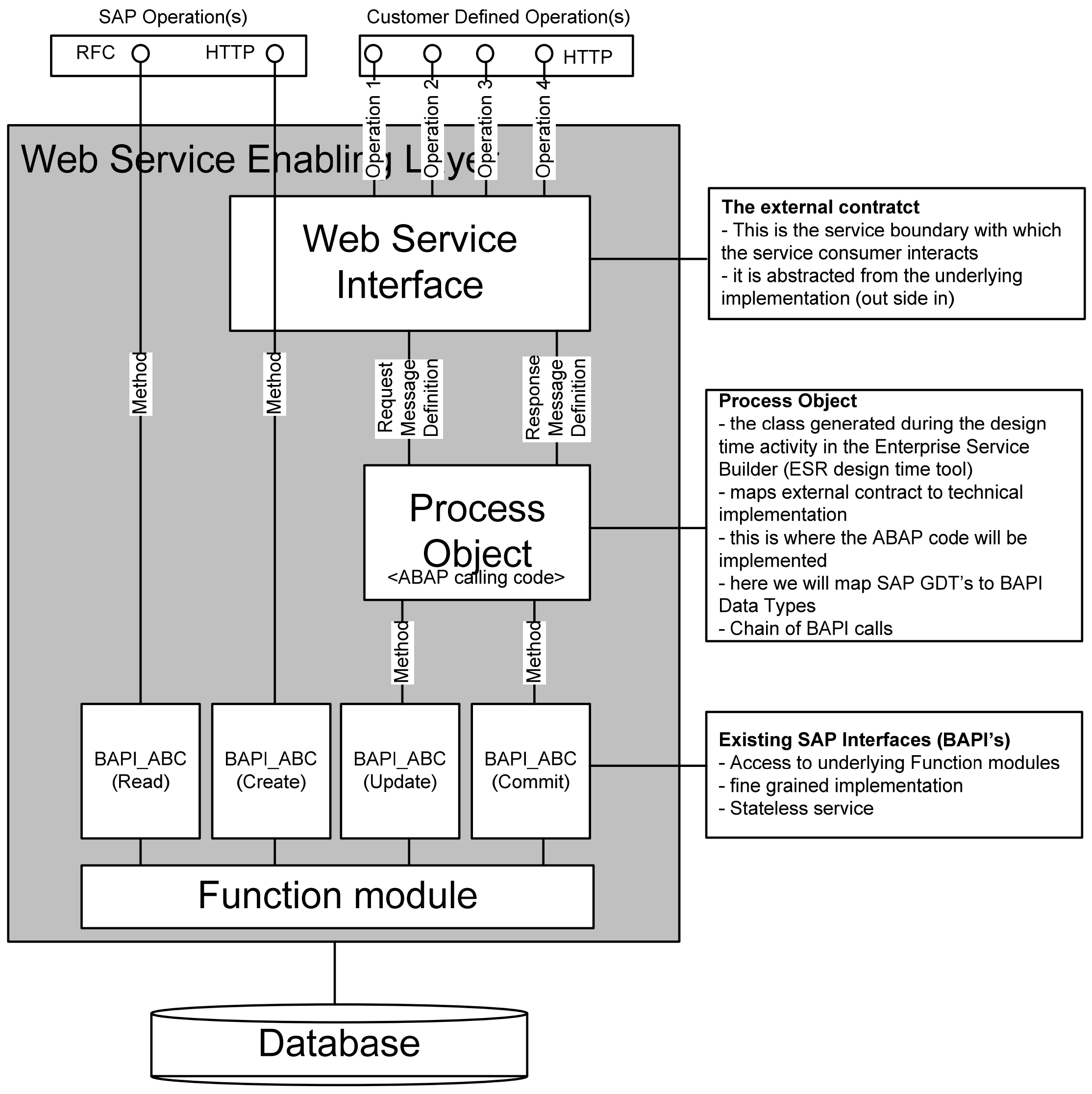
- Customer defined **data type to be used in conjunction with any of the available communication protocols.** As described earlier in the document, this option would force tight coupling between the project business service data model and SAP functionality.
- **ABAP dictionary data types with Web Services.** Modelling Business Objects using ABAP dictionary data types will force adoption of non standard content and result in a custom implementation and increased solution cost (maintenance, upgrade, testing etc.)

As an approach, **Web Services** (Enterprise Service from SAP where defined) should be preferable over RFC. This is the strategic approach from SAP and new functionality (interface plus transactions) will be delivered using 'services' only.

Over time the amount of pre-defined Enterprise Services from SAP will increase, and in line with the performance information available for scenarios and implementation will also be made 'standardised' and visible.

To change this approach the benchmarks should indicate a significant difference between the two options that will breach performance budget created for the components in the landscape and the end to end performance threshold for the given scenario.

Possible implementation options are:


SAP Operation(s)
RFC
HTTP
Customer Defined Operation(s)
HTTP
Operation 1
Operation 2
Operation 3
Operation 4
Web Service Enabling Layer
Web Service Interface
Method
Method
Request Message Definition
Response Message Definition
Process Object
<ABAP calling code>
Method
Method
BAPI_ABC (Read)
BAPI_ABC (Create)
BAPI_ABC (Update)
BAPI_ABC (Commit)
Function module
Database
The external contratct
- This is the service boundary with which the service consumer interacts
- it is abstracted from the underlying implementation (out side in)
Process Object
- the class generated during the design time activity in the Enterprise Service Builder (ESR design time tool)
- maps external contract to technical implementation
- this is where the ABAP code will be implemented
- here we will map SAP GDT's to BAPI Data Types
- Chain of BAPI calls
Existing SAP Interfaces (BAPI's)
- Access to underlying Function modules
- fine grained implementation
- Stateless service

Enterprise Services with Global Data Types

Pros:

- This option aligns well with the future SAP strategy
- Content is being delivered by SAP to 'adopt' within a project with no SAP specific modelling effort
- Web Services expose the SAP functionality through an industry standard interface
- This option provides all the inherent benefits of SOA
- Exposing SAP functionality through web services will allow choices to be made around consumption, and ensure service consumers are abstracted from the underlying implementation on SAP and isolated from the technical changes that will be required.
- Web Services expose stateless functionality
- Web Services interface support guaranteed message delivery via Web Services Reliable messaging and idempotency.

Cons:

- Even though this is SAP's strategic direction and recommendation, the definitions and available Enterprise Services for the banking functionality are not complete.
- The GDT's available from SAP are not complete and will need to be extended to accommodate data types not currently delivered.
- This option will have slower runtime as compared to other options since it adds the runtime overhead of not only the web dispatcher (web services proxy layer) but also the extra transformation required to transform data types.

Web Services Outside-In using Global Data Types

This option would require using the Enterprise Services Builder tooling to create, release and manage services into the environment. There will be an overhead of modelling effort to correctly import (create /extend) the data types within the namespace that you are deploying services.

The outside in modelling will however allow abstraction away from the underlying implementation, create services and operations at the correct level of granularity, and ensure that the data types used are managed in line with the tooling and strategy from SAP.

Web Services Inside-Out with existing SAP data types

Pros:

- Provides access to existing interfaces through a standard message format and transport.
- By using the native ABAP data types, this option avoids any extra data transformation other than transforming external data types to ABAP dictionary data types.

Cons:

- The operations from the underlying RFC implementation are limited, which drives customisation of the exposed interface to ensure correct granularity is exposed – BAPIs expose function modules (single operation) and as such are 'atomic' transactions.
- WS interface is just being used as integration mechanism and will cause versioning and upgrade issues – underlying project specific function module changes will require every interface to be re-deployed at upgrade.
- Limited tooling support other than 'traditional' BAPI release tooling (check box to release as 'web service').

RFC/ BAPI with SAP ABAP dictionary data types

Before the advent of Web Services, RFC was the SAP predominant integration mechanism.

Pros:

- Provides the least processing overhead when exposed at the lowest level of granularity. Therefore, this may be a suitable option for uses cases (e.g. Payments) that have limited operations and stringent performance requirements.
- Avoids any extra transformation required for transforming abstracted data types (GDT's) outside the SAP environment to the SAP data types.
- RFC supports guaranteed delivery, stateful sessions and transactional integrity, if it is required. However, this does not comply to the notion of a web service.

Cons:

- RFC is a proprietary protocol and is not an industry standard.
- This option is not aligned with SAP's strategic direction of exposing functionality through Enterprise Services and GDTs.
- RFC's provide only single operations based on SAP functions – there is a composition/orchestration overhead to consume RFC's at the correct level of granularity, typically middleware tooling is used.
- BAPI's provide stateful operations. Transactional integrity and in most cases idempotency has to be managed in a middleware layer or has to implemented 'on top' of the existing BAPI's in the ABAP stack.

Possible Discussion Points and Rules of Thumb in Implementation Projects

Besides functional like granularity and non-functional discussion points like transactional integrity, idempotency, security and performance, we would like to mention the following ones we already faced in implementation projects. Rules of thumb will be given what should be considered for a selection of the integration approach.

Possible discussion points

Inside-out versus outside-in

- Using Web Services with inside-out approach – exposing existing functionality as a web service while trying to create a reasonable way to cluster (or categorise) the services, results in an effort to map.
- The usage of Enterprise Services forces an outside in approach: a drill down from the consumer process (context) to the functionality exposed. The modelling is an additional effort, but, it also results in an SAP-aligned modelled layer.

GDTs transformation to ABAP dictionary

- The use of GDT requires an additional transformation to be done in the ABAP class level to ABAP dictionary. Although this additional layer would result in performance overhead, this is a hard coded layer that does not use any transformation mechanism due to its static nature, thus, adding only a fraction of CPU effort to the whole process.
- Regarding development effort, we should take in account that for GDTs that are already exposed by an existing Enterprise Service we will usually find an ABAP class that implements the transformation to the ABAP data types. If this option is chosen, with time, the additional GDTs will have a local implementation of transformation, thus, reducing the effort for each new interface.

SAP roadmap and strategy for Enterprise Services

- The SAP roadmap states that, while Enterprise services exposed from the SAP business objects are the strategy, SAP will not be removing support for existing BAPI's in its solution.
- As the platform evolves, more Enterprise Services will be made available. There will be less investment made to develop and implement new BAPI's as an external interface type because of this focus.
- This in line with SAP's support strategy for existing solutions deployed using existing technology (interface) types.

Rules of Thumb for Consideration

Integration approaches should be evaluated for each integration scenario (e.g. payments, channel applications, etc.). Guiding principles for the implementation should be determined according to these evaluations.

Recommendations could be based on the following areas:

- Communication sequence
- Process orchestration
- Granularity of service functionality
- Number and complexity of service operations
- Transactional Integrity
- Idempotency
- Security aspects

- Infrastructure (e.g. load balancing)
- Performance requirements
- Environment considerations (e.g. integration end-point exists already to consume web services etc.)

Below rules of thumbs are given for chosen integration approaches.

Scenario: Adopt BAPI based interfaces (Interface mapping only)

(Inside-out: Use the existing BAPI with its single method as the exposed interface)

Pro:

- For speed of deployment and matching
- Building on existing interfaces and existing mapping
- Skills are readily available
- Allows flexibility in exposure (choice of SOAP HTTP or RFC Binary)
- Allows project to build a migration path from BAPI to Web Service at their own time frame
- Reduces amount of additional infrastructure needed
- Ensures project scales are achieved without risk of late delivery from incomplete modelling (mapping only approach)

Con:

- Not aligned to the SAP strategic direction of web services
- Does not allow for modelling using SAP tools (Enterprise Services Builder)
- Close coupling of consumer (e.g. channel systems) to provider (SAP)
- Proprietary interface types
- Poor security model
- HTTP load balancing available for binary connections

Rule of Thumb:

For scenarios that meet the following criteria BAPI (RFC over binary) is recommended

- Closely coupled interface to consuming application for performance
- Application interface is not able to ‘change’
- Security is not primary concern

Scenario: Migrate BAPI interfaces to Web Services

(outside-in: Model the interface then map to the underlying BAPI implementation)

Pro:

- Allows managed migration to strategic interface technology
- Will leverage additional content that is scheduled possible future releases
- Allows the services landscape to be migrated based on actual utilisation not modelled services
- Ensure re-use of interfaces based on harmonised service landscape

Con:

- Lack of detailed enterprise services release schedule
- Additional modelling effort to create web service definitions
- Harmonisation effort to ensure message semantics are correctly aligned (Customer <-> SAP)

Rule of Thumb:

For scenarios that meet the following criteria Web Services is recommended:

- Re-useable interface with multiple consumers
- Web service enabled application
- Requires security over and above simple user name password, e.g. SAML
- Where strong Web Service standards exist (non-proprietary) for message type and transport
- Where performance is not primary concern

Scenario: Adopt SAP Enterprise Services: harmonised semantics (data and structure)

(SAP defined outside-in: Adopt Enterprise Services as the de-facto implementation

Pro:

- Allows managed migration to strategic interface technology
- Will leverage additional content that is scheduled for future project releases
- Allows the services landscape to be migrated based on actual utilisation not modelled services
- Ensure re-use of interfaces based on harmonised service landscape

Con:

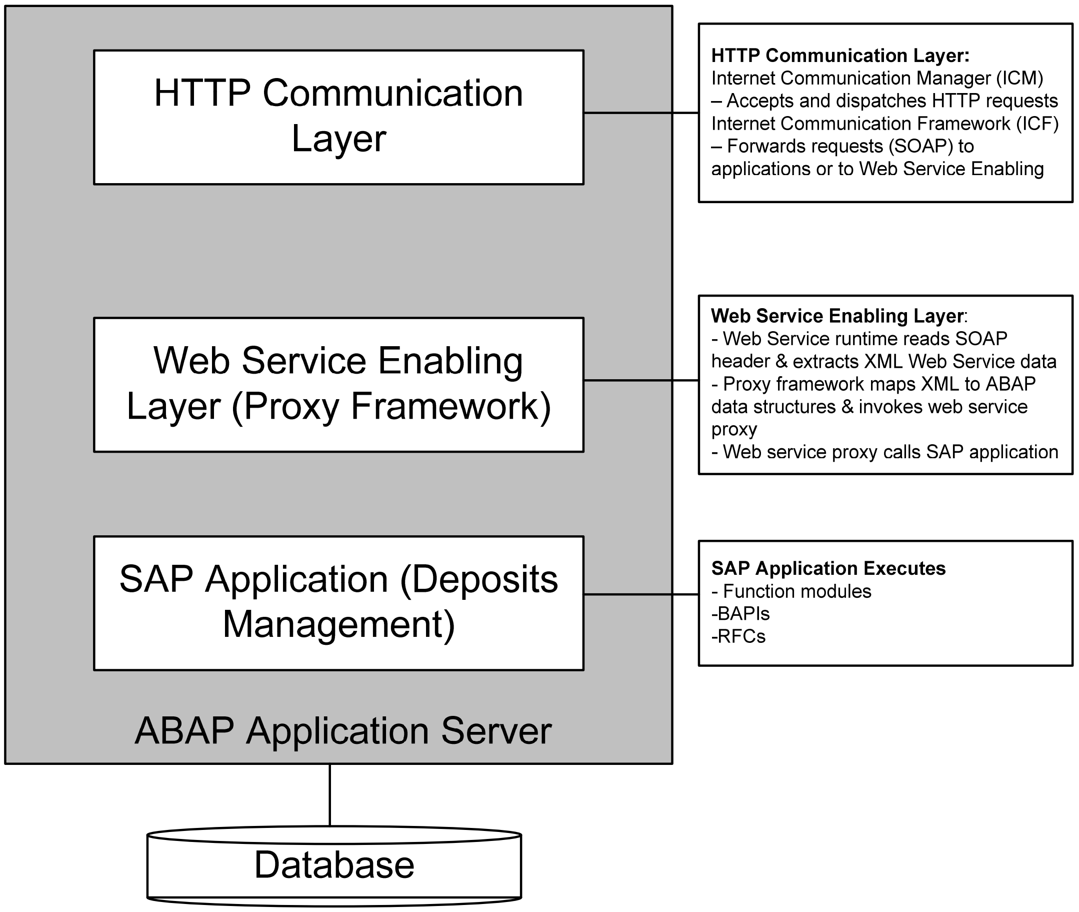
- Lack of enterprise services release schedule
- Harmonisation effort to ensure message semantics are correctly aligned (Customer <-> SAP)

Rule of Thumb:

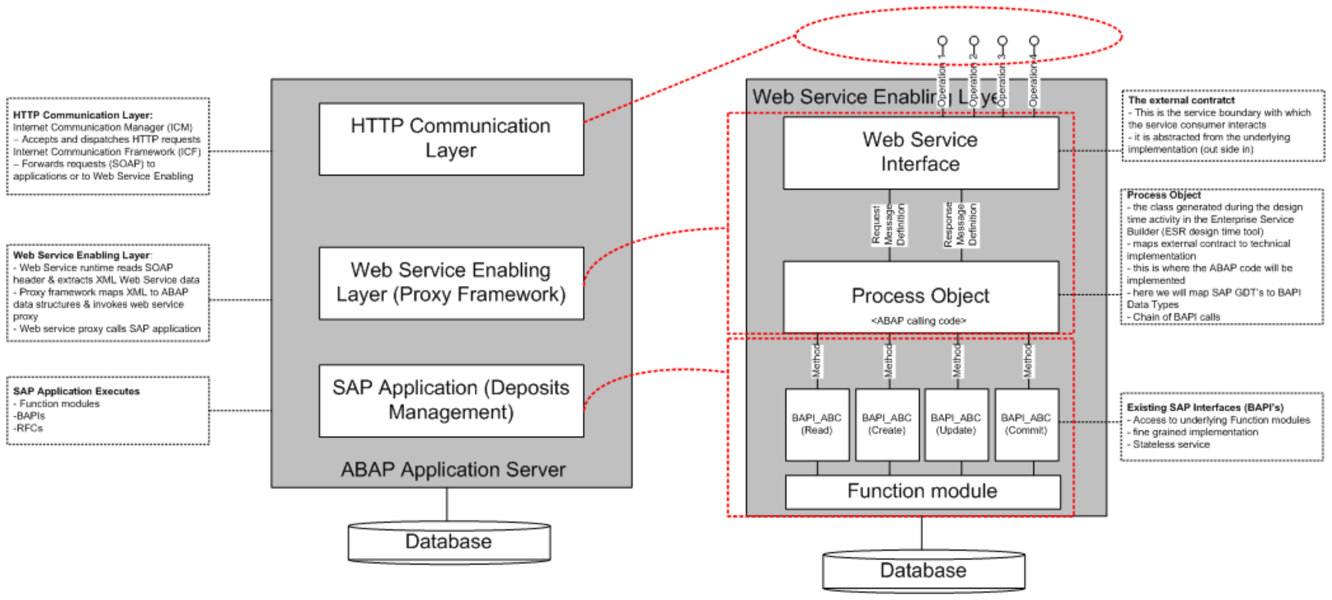
Use Enterprise Services whenever possible, especially

- For scenarios that require strategic stability
- for scenarios that can leverage existing SAP content and are semantically aligned
- for consumer applications that can align customer Data model to SAP Data types
- for consumer applications that can align customer message types to SAP message types

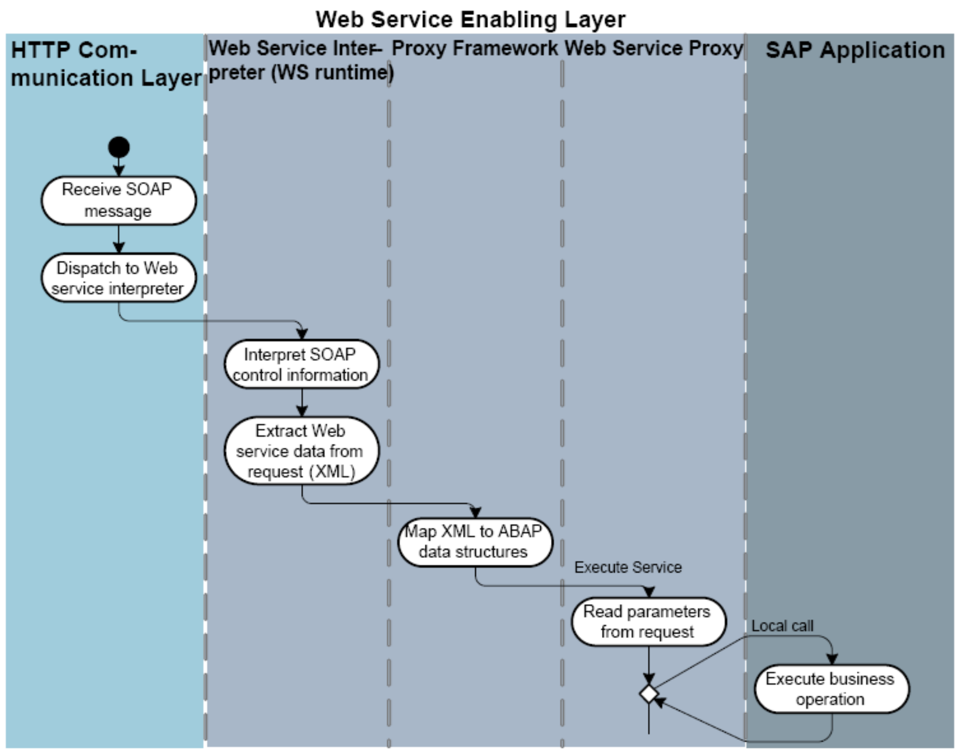
Appendix 1: RFC vs. WS Performance comparison


HTTP Communication Layer
Web Service Enabling Layer (Proxy Framework)
SAP Application (Deposits Management)
ABAP Application Server
Database
HTTP Communication Layer:
Internet Communication Manager (ICM)
– Accepts and dispatches HTTP requests
Internet Communication Framework (ICF)
– Forwards requests (SOAP) to applications or to Web Service Enabling
Web Service Enabling Layer:
- Web Service runtime reads SOAP header & extracts XML Web Service data
- Proxy framework maps XML to ABAP data structures & invokes web service proxy
- Web service proxy calls SAP application
SAP Application Executes
- Function modules
-BAPIs
-RFCs

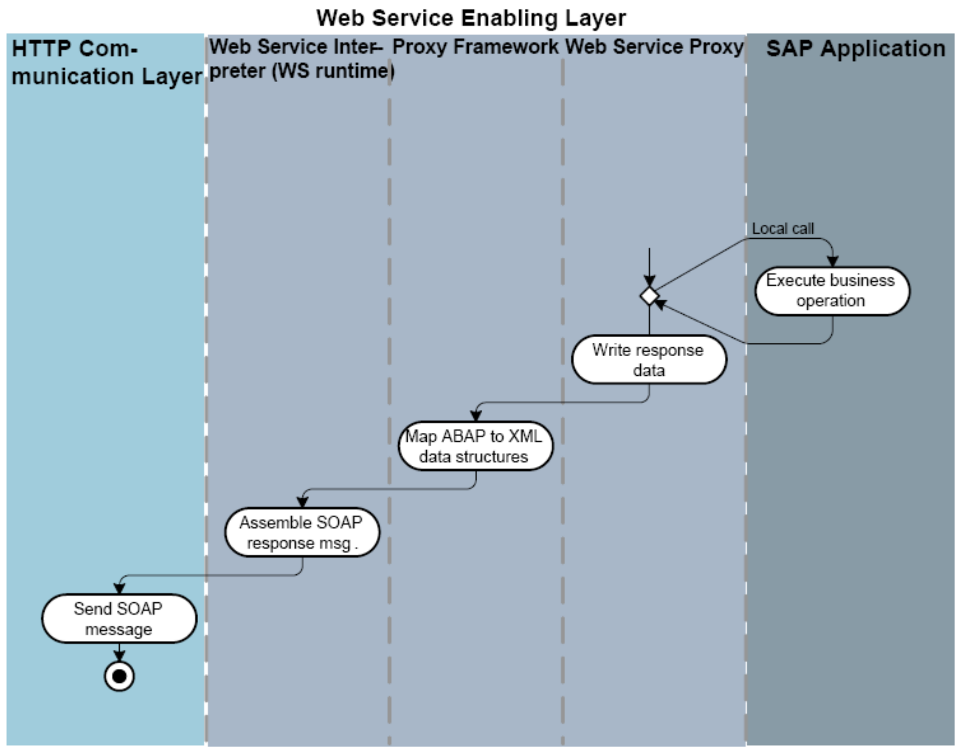
IN order to explain the layers the following diagram is also of use:


HTTP Communication Layer:
Internet Communication Manager (ICM)
– Accepts and dispatches HTTP requests
Internet Communication Framework (ICF)
– Forwards requests (SOAP) to applications or to Web Service Enabling
Web Service Enabling Layer:
- Web Service runtime reads SOAP header & extracts XML Web Service data
- Proxy framework maps XML to ABAP data structures & invokes web service proxy
- Web service proxy calls SAP application
SAP Application Executes
- Function modules
-BAPIs
-RFCs
HTTP Communication Layer
Web Service Enabling Layer (Proxy Framework)
SAP Application (Deposits Management)
ABAP Application Server
Database
Operation 1
Operation 2
Operation 3
Operation 4
Web Service Enabling Layer
Web Service Interface
Request Message Definition
Response Message Definition
Process Object
<ABAP calling code>
Method
Method
Method
Method
BAPI_ABC (Read)
BAPI_ABC (Create)
BAPI_ABC (Update)
BAPI_ABC (Commit)
Function module
Database
The external contratct
- This is the service boundary with which the service consumer interacts
- it is abstracted from the underlying implementation (out side in)
Process Object
- the class generated during the design time activity in the Enterprise Service Builder (ESR design time tool)
- maps external contract to technical implementation
- this is where the ABAP code will be implemented
- here we will map SAP GDT's to BAPI Data Types
- Chain of BAPI calls
Existing SAP Interfaces (BAPI's)
- Access to underlying Function modules
- fine grained implementation
- Stateless service

The following diagram shows the flow between the logical components above:


Web Service Enabling Layer
HTTP Com-munication Layer
Web Service Inter-preter (WS runtime)
Proxy Framework
Web Service Proxy
SAP Application
Receive SOAP message
Dispatch to Web service interpreter
Interpret SOAP control information
Extract Web service data from request (XML)
Map XML to ABAP data structures
Execute Service
Read parameters from request
Local call
Execute business operation

And the corresponding return route:


Web Service Enabling Layer
HTTP Com-munication Layer
Web Service Inter-preter (WS runtime)
Proxy Framework
Web Service Proxy
SAP Application
Local call
Execute business operation
Write response data
Map ABAP to XML data structures
Assemble SOAP response msg .
Send SOAP message

Relative response times based on local connectivity: the difference in synchronous RFC (sRFC) and Web Service is nominal. In turn the relative performance difference in all communication types converges as the overall application runtime increases.

Scenarios and integration mechanisms should always be tested within context of platform, consumer and provider implementations. In this way end to end performance budgets[7] can be created that allow a project to deliver suitable interfaces given the operational requirements of the solution.

Appendix 2: Global Data Types

Why use data types?

You use data types to describe the data structure of a message type that can be referenced from one or more service interfaces in the ES Repository. By using the service interface, you can use the referenced data structure for the following scenarios:

- To define the data of a message that is exchanged in either Web service communication (point-to-point) or in
- Integration Server communication. The data type definition determines how the valid payload of a message will look.
- To use the data of an (adaptive) Web service model in the SAP NetWeaver Developer Studio for UI development or generally in the SAP Composition Environment including Business Process Management. The Web service model is based on a Web service definition in the ABAP backend, which in turn you can create on the basis of ES Repository service interfaces in the backend.

What Data Types are available?

The following classifications of data types are available in the ES Repository:

- Core data types (internationally standardized data types that are based on W3C data types) and aggregated data types (combining several core data types)[8].
- Freely-modelled data type – they support XML schema only. More information about data types in the ESR[9].

The message types in the ES Repository can reference all data types: freestyle data types, core data types, and aggregated data types.

Based on these classifications, the following GDT categories are available:

- Basic GDTs – based on core data types (for example, the PersonID GDT is based on the core data type Identifier).
- Aggregated GDTs (Complex GDTs) – combine several basic GDTs together to define a complex business-related subject matter (for example, Address combines the basic GDTs StreetName, StreetNumber, and so on)

You can develop the following data types:

- CCTS-based core data types10
- CCTS-based aggregated data types11

---

[7] https://www.sdn.sap.com/irj/sdn/index?rid=/webcontent/uuid/10986f61-0a01-0010-bdad-cdd93247dbc1

[8] http://help.sap.com/saphelp_nwpi71/helpdata/en/45/614fc4c5293bdce10000000a1553f7/content.htm

[9] http://help.sap.com/saphelp_nwpi71/helpdata/en/45/607248b5b33bdbe10000000a1553f7/content.htm

[10] Developing CCTS-based core data types:

http://help.sap.com/saphelp_nwpi71/helpdata/en/96/5f0e450de94202b39d984de2428cce/frameset.htm

[11] Developing CCTS-based aggregated data types:

- Simple freely-modeled data types12
- Complex freely-modelled data types13

How can I reuse global data types?

After you model the application services and the applications' interactions with other applications, you can proceed with a detailed definition of the modelled design objects, such as the operations in the service interface and the data structures that describe a particular message type.

Business objects describe data of a well-defined and well-outlined business area. The structure of this data is defined using data types in the Enterprise Services Repository.

To achieve a high level of reusability for data types, SAP uses global data types (GDTs). GDTs are SAP-wide normalized and reconciled data types with business-related content and are based on standards. GDTs ensure harmonization of the hierarchical signature structures of operations and the structure of the business object nodes. They represent a business-related subject matter that is described by a specified structure.

If this semantic subject matter occurs in a business object node or a B2B or A2A operation, it is always typed by the same global data type. This leads to uniform typing across all business objects, interfaces, and operations. GDTs have the following characteristics:

- Reusable semantic building blocks for service operations and business objects.
- Based on international standards (ISO 15000-5 and UN/CEFACT CCTS).
- SAP-wide approved according to the overall SAP governance process for business content.
- Defined in the ESR and described by XML schema.
- Well documented in accordance with the documentation templates.

For more information about global data types, refer to the **Data Type Catalog**[14].

How do I define own global data types?

GDTs can be provided context-specifically and with more semantics regarding a business-related context data types when used in operations (or business objects). The elements used can also be restricted and integrity constraints can be made stricter. Thus, based on a maximal GDT, further data types can be projected as context-specific restrictions.

Creating New Data Types

You can develop CCTS-compliant data types or freely-modelled data types. You can also reference core data types or aggregated data types from freely-modelled data types. SAP recommends that you develop CCTS-based data types because this way you increase the likelihood that the data types can be reused for the development of A2A and B2B processes, as well as on the UI. You find the CCTS-specific attributes and properties of this methodology in the data type editor in the Enterprise Services Builder. However, developing data types that comply with CCTS requires governance committees in one or more companies. CCTS only forms the basis of these processes because it enables the meaning of data types and their attributes to be represented in the same way by means of naming and structure conventions.

---

http://help.sap.com/saphelp_nwpi71/helpdata/en/cf/e0a6319e064f5c8ef3da4fa00cc269/frameset.htm

[12] Developing simple freely-modeled data types:

http://help.sap.com/saphelp_nwpi71/helpdata/en/45/619ec603e61feee10000000a1553f7/frameset.htm

[13] Developing complex freely-modeled data types:

http://help.sap.com/saphelp_nwpi71/helpdata/en/45/6075feb5b33bdbe10000000a1553f7/frameset.htm

[14] Data Type Catalog: https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/303fd192-1db2-2a10-59b9-9dfd93f4b10f55

For more information, refer to the UN/CEFACT Techniques and Methodologies Group pages[15]

Enhancing Data Types

If you need to enhance the SAP applications source code, you can do so by using Business Add-Ins (BAdIs) and therefore there is no need to modify SAP applications directly. In addition, you can define data type enhancements for data types for classifying freely-modelled data types or aggregated data types in the ESR without modifying the SAP standard. If the service interfaces developed in the ESR are to be used for exchanging messages, you can use these enhancements to transfer additional data with a message, which can then, for example, be accessed using a BAdI[16].

SAP Global Data Types (GDT) Definition:

- Re-usable data types for service interfaces and business object nodes.
- Approved SAP-wide by the Governance Process for Business Content (advanced by Process Integration Council (PIC))
- Represent a business-related subject matter described by a specified structure.
- Maximally defined data types containing all elements required for the subject matter in different contexts.
- Are basis for further data types which are derived as context-specific restrictions.
- Are based on the data type development methodology described in the international Standards ISO 15000-5 and UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) CCTS (Core Component Technical Specification).
- GDTs are – according to their nature – defined usage neutral, but with business semantics.
- Defined in ES Repository using Extensible Mark-up Language (XML) schema

Within SAP standard development, it is mandatory to use GDTs to define business object attributes and service interface parameters. With this approach, SAP ensures that if the same attribute occurs in business object nodes or service interfaces, it is always described by the same or a derived GDT.

[15] UN/CEFACT pages:

http://www.untmg.org/index.php?option=com_docman&task=view_category&Itemid=137&subcat=14&catid=65&limitstart=0&limit=5

[16] Enhancing data types:
http://help.sap.com/saphelp_nwpi71/helpdata/en/45/61aaa95e0b2073e10000000a1553f7/frameset.htm

Copyright