

Banking Industry Architecture Network

BIAN **How-to Guide** **Semantic API**

Organization

Authors		
Role	Name	Company
BIAN Architect	Guy Rackham	BIAN

Status			
Status	Date	Actor	Comment / Reference
DRAFT	October 2018		Revised for Review
Approved		Architectural Committee	

Version		
No	Comment / Reference	Date
7.0	First edited version	October 2018

Copyright

© Copyright 2018 by BIAN Association. All rights reserved.

THIS DOCUMENT IS PROVIDED "AS IS," AND THE ASSOCIATION AND ITS MEMBERS, MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THIS DOCUMENT ARE SUITABLE FOR ANY PURPOSE; OR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE ASSOCIATION NOR ITS MEMBERS WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THIS DOCUMENT UNLESS SUCH DAMAGES ARE CAUSED BY WILFUL MISCONDUCT OR GROSS NEGLIGENCE. THE FOREGOING DISCLAIMER AND LIMITATION ON LIABILITY DO NOT APPLY TO, INVALIDATE, OR LIMIT REPRESENTATIONS AND WARRANTIES MADE BY THE MEMBERS TO THE ASSOCIATION AND OTHER MEMBERS IN CERTAIN WRITTEN POLICIES OF THE ASSOCIATION.

Contents

1. Introduction	6
2. BIAN Design Principles.....	7
2.1 - BIAN Service Domain Architectural Properties	8
2.2 - Business Role/Scope of Activity of a Service Domain	9
2.3 – Internal Operation of a Service Domain	12
2.4 – Service Based Access.....	13
2.5 – Defining Message Content – The BIAN API Platform Sandbox	15
3. Key Benefits of using BIAN to design API solutions	17
3.1 Support for Emerging Industry Approaches	17
3.2 Support for Industry Standards	19
3.3 Support for Incremental Adoption/Migration.....	20
4. Using BIAN as an API ‘Inventory’.	22
5. Three types of architectural alignment	25
5.1 Direct to Core	25
5.2 Wrapped Host	27
5.3 Distributed Architecture	29
6. BIAN Design elements used in API specification.	32
6.1 Extended Service Domain Specifications.....	32
6.2 Wireframes (showing enterprise boundaries)	33
6.3 Business Scenarios.....	35
6.4 Service Operations.....	36
7. BIAN API Design Approach	38
7.1 General Implementation Considerations	38
7.1.1 Semantic API Selection Framework.....	39
7.1.2. Service Domain Cluster	42
7.2 For Type 1 – Direct to Core	43
7.3 For Type 2 – Wrapped Host.....	44
7.4 For type 3 – Distributed Architecture	45
8. BIAN Design Content – SL V 7.0	49
Attachments - Example Wave 1 Deliverables	51
The wireframe	51
Business Scenarios	52

Figure 1: Simple Schema of a Service Domain Design.....	10
Figure 2: Functional Patterns & Their Generic Artifacts.....	10
Figure 3: Value Chain Service Landscape Layout	11
Figure 4: Functional Patterns, Generic Artifacts and Behavior Qualifier Types.....	12
Figure 5: Service Domain Behavior Qualifiers Added to Service Operations	13
Figure 6: BIAN Action Terms	13
Figure 7: Default Action Term By Functional Pattern	14
Figure 8: BIAN Service Domains Related to Micro-services	19
Figure 9: The Service Landscape with Open API Candidates.....	23
Figure 10: Example Wave 1 – Service Operation Descriptions	23
Figure 11: Summary of API Sophistication Levels	25
Figure 12: Type 1 Layout	26
Figure 13: Type 2 Layout	28
Figure 14: Type 3 Layout	30
Figure 15: Expanded Service Domains (Excel).....	33
Figure 16: Wireframe Example from Wave 1	34
Figure 17: Mobile Access Wireframe with Time Dependencies.....	35
Figure 18: Example Business Scenario.....	36
Figure 19: Service Operation Definition.....	37
Figure 20: Simplified API Technique Framework	39
Figure 21: List of 28 Techniques Under the 4 Categories	41
Figure 22: Application Cluster Example	43
Figure 23: Type 3 With Contact Points Highlighted	46
Figure 24: Type 3 - Expanded Customer Access Platform.....	47
Figure 25: API Content Development Approach.....	49
Figure 26: Wireframe example - Customer offers / onboarding.....	51
Figure 27: Prospect On-boarding (KYC).....	52
Figure 28: Suitability/Eligibility Checks	52
Figure 29: Configuration and Pricing Negotiation	53
Figure 30: Credit/Risk/Underwriting Decisions	53
Figure 31: Documentation & Compliance Checks	54
Figure 32: Product Booking, Recording and Set-up Initiation	54

1. Introduction

This is the BIAN Semantic Application Programming Interface (API) How to Guide. It is being released in conjunction with the BIAN Service Landscape Version 7.0 (SLV7.0). The BIAN SLV7.0 release includes significant content extensions that support the use of the BIAN Service Landscape as an API solution directory and BIAN specifications as high-level designs for standards-based API development. The Service Landscape release has been coordinated with the launch of the BIAN API Platform Sandbox – an open source developer environment that presents RESTful endpoint definitions for a selection of BIAN Service Domain as outlined in more detail in this guide.

The intended audience for the guide is business and technical architects tasked with the development and deployment of API ecosystems. It assumes a basic understanding of the BIAN design principles and content. A brief summary of the BIAN approach is included at the beginning of this guide to provide background. The guide is set out as follows:

- **BIAN Design Principles** – an overview of key BIAN design principles and techniques as they apply to API design
- **Key benefits of using BIAN** – the BIAN design properties that support API design
- **BIAN Service Landscape API Inventory** – the BIAN SL can be used to classify and inventory available ‘open’ APIs
- **Three Levels of alignment** – The full adoption of a componentized model is likely to be a migration for most participants – three distinct levels of alignment are defined
- **BIAN API design elements** – descriptions and examples of the BIAN design artifacts used in semantic API specification
- **BIAN API approach** – some general solution development techniques and then adoption approaches specific to each of the three defined levels of BIAN alignment
- **An overview of BIAN API “Wave 1” content** – BIAN API aligned specifications published with BIAN SLV7.0

The BIAN model of banking adheres to key principles by defining discrete ‘Service Domains’ that perform standard (canonical) business roles. The BIAN Service Domains interact through service operations and these service operations can be used to outline the business purpose and high-level information content of APIs where appropriate. The interaction of service domains and service operations represent information flowing through an organization.

2. BIAN Design Principles

The BIAN design principles are fully described in the BIAN How To Guides published with each release of the Service Landscape. A summary of the main design concepts underlying the BIAN standard is provided here with emphasis on how the BIAN design approach supports API specification for reference.

Different Model Perspective

BIAN defines a particular model of banking activity. As opposed to more traditional process oriented views of business, BIAN captures banking activity by first isolating discrete partitions of business functionality that can then be employed in any suitable combination to address any business event/need. These business capability 'building blocks' collaborate with each other through service operation exchanges. BIAN calls these building blocks Service Domains.

An example clarifies the distinction between a more traditional process and a BIAN view of business. Consider a bakery: a conventional process view would describe the recipe and steps to follow when baking a cake (as one might find set out in a cookbook). Conversely the BIAN view would first describe the different things found in the bakery: the food ingredients, cooking utensils and kitchen equipment and the actors involved (the chef). These items define 'static' properties/ingredients of the bakery. BIAN then represents the event of baking a cake as a pattern of collaboration between a suitable selection of these parts which is a 'dynamic' model capturing a particular supported behavior of the bakery.

Both views are useful but are suited to defining very different types of systems solutions. The process representation is used to define the procedure to follow to bake a cake that is consistent/repeatable. It can be streamlined/made more efficient and perhaps is suited to automation in parts. The BIAN representation is used to define the range and key properties of required components (Service Domains) and then represent different views of how these components may be reused in different combinations to meet different needs. The BIAN model is good to ensure each component is 'fit for purpose' and with the flexibility to capture how they can be reused in many different combinations, in this example to bake a cake but potentially in other ways to prepare different types of food.

BIAN Service Domains

The design concepts underlying the BIAN standard are quite complicated and it can take some time and practical experience to become fully conversant in the BIAN approach. But as noted, they define a representation that is particularly well suited to the componentized and highly distributed systems architectures that APIs support.

Here the BIAN concepts are outlined working top-down from the definition of the Service Domain itself. These descriptions should be considered along with the 'Wave 1' examples presented later in this guide:

1. *BIAN Service Domain Architectural Properties* - first the general definition of a Service Domain is provided in terms of the main design principles
2. *Business Role/scope of Activity of a Service Domain* - then the way the business role of a Service Domain is scoped out is explained
3. *Internal Operation of a Service Domain* - next the internal workings are outlined to reveal how the business functions and information governed by a Service Domain is identified
4. *Service Based Access* - describes the way a Service Domain's business functions and information are accessed through offered services
5. *Determining Message Content* - finally the way the content of the underlying service operation messages is selected from the overall information governed by the Service Domain is considered

2.1 - BIAN Service Domain Architectural Properties

A BIAN Service Domain is a conceptual business design that defines a partitioned or componentized view of business behavior that has the following key properties:

- Each Service Domain supports a unique and discrete (non-overlapping) business role or purpose – *for example designing products, operating an ATM network, authenticating a customer, handling a customer contact...*
- The Service Domain handles the execution of its business role for the full life cycle (from start to finish) as many times and for as many concurrent activations as may be needed – *for example creating and maintaining a single corporate strategy, or setting-up and maintaining 30 million active customer contracts*
- The business role must also be 'elemental' in nature i.e. uniquely assignable to a single responsible party in the organization. The execution of any business role will typically break down into many finer grained tasks/steps but these will be utility/non 'role specific' in nature – *for example customer relationship management is a uniquely assignable role, but the underlying activities involved such as holding customer meetings, developing plans/budgets and analyzing contact records may re-occur in many different business contexts other than relationship management.*
- The Service Domain's business role/purpose must be canonical (i.e. be consistently interpretable when implemented in different situations by different organizations) – *for example the need to authenticate a customer can be consistently defined (how it is done in practice can vary place to place and also evolve over time. But the business requirement to be able to authenticate customer is unequivocal/canonical)*

- Service domains can be architected as shared services reducing operating complexity and costs with re-used solution components and enabling a migration from monolithic legacy solutions dedicated to single lines of business or functional departments of the organization.

BIAN Service Domains are typically assumed to interact using service operations as they collaborate in any appropriate combination and sequence in the execution of business. When using BIAN to develop API designs the relevant Service Domains are mapped/associated with business applications and their external boundaries. The service operation exchanges then outline the key business purpose and information content of external interactions as such defining a high-level specification of an application program interface (API).

2.2 - Business Role/Scope of Activity of a Service Domain

In order to define an industry standard for service based design a key challenge for BIAN was to find a way to scope out Service Domains that are truly canonical. The technique BIAN employs includes a two-step procedure to identify candidate Service Domains. The candidate roles are then tested out in practice using real world banking scenarios to refine their definitions. The technique is as follows:

- First a notional 'bank' has been decomposed into the collection of all possible tangible and intangible 'assets' that might make up the bank under the assumption that all banks are assembled from similar things – *for example computer networks, buildings, product delivery capacity, servicing centers, customer and worker relationships, market knowledge and product know-how*
- Second, different behaviors have been identified that are used to foster and exploit these assets for commercial gain. These commercial behaviors are called Functional Patterns – *for example 'operating' computer networks, 'maintaining' buildings, 'fulfilling' products/services, 'managing' relationships, 'analyzing' markets. BIAN has rationalized these behaviors to identify 18 distinct Functional Patterns that cover all types of commercial activity*

A Service Domain applies **one** of the 18 identified functional patterns to instances of **one** type of asset for the full life cycle of that asset's use and for as many times as is necessary. A mechanism called a '*control record*' is used to track the life cycle state each time the Service Domain performs its role.

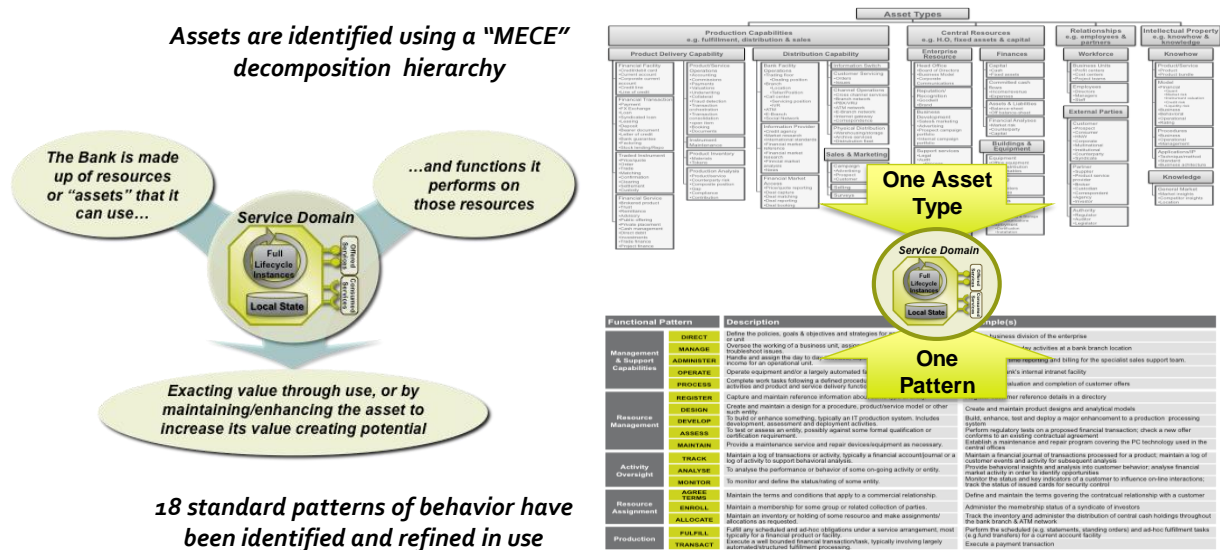


Figure 1: Simple Schema of a Service Domain Design

Depending on the functional pattern instances of this control record can be characterized as being a type of artifact. BIAN has defined 'generic artifacts' corresponding to each of the standard functional patterns. The table below lists the standard Functional Patterns and their associated generic artifacts:

Functional Pattern	Generic Artifact
DIRECT	Strategy
MANAGE	Management Plan
ADMINISTER	Administrative Plan
OPERATE	Operating Session
PROCESS	Procedure
REGISTER	Directory Entry
DESIGN	Specification
DEVELOP	Development Project
ASSESS	Assessment
MAINTAIN	Maintenance Agreement
TRACK	Log
ANALYSE	Analysis
MONITOR	Measurement
AGREE TERMS	Agreement
ENROLL	Membership
ALLOCATE	Allocation
FULFILL	Fulfillment Arrangement
TRANSACTION	Transaction

Figure 2: Functional Patterns & Their Generic Artifacts

A service Domain's control record is defined to be the combination of the asset type acted upon and the generic artifact corresponding to its Functional Pattern. Additional explanation as to how BIAN defines Service Domains can be found in the BIAN How To Guide – Design Principles & Techniques.

At this stage by modeling real world scenarios with its membership BIAN has isolated about 300 discrete Service Domains representing valid/practical combinations of an asset type and functional pattern. The Service Domains are organized in a simple reference architecture called the BIAN Service Landscape.

BIAN maintains two layouts of the Service Landscape, a 'matrix' layout the format of which dates back to the early days of BIAN and a more recently developed 'value chain' layout shown below. Both include all of the identified BIAN Service Domains. The difference between the two views is only the layout and scope of larger Business Areas and Business Domains that are simply used to organize the Service Domains for ease of reference. For the API work the value chain layout has been adopted as this corresponds more closely to the typical organizational elements of a Bank.

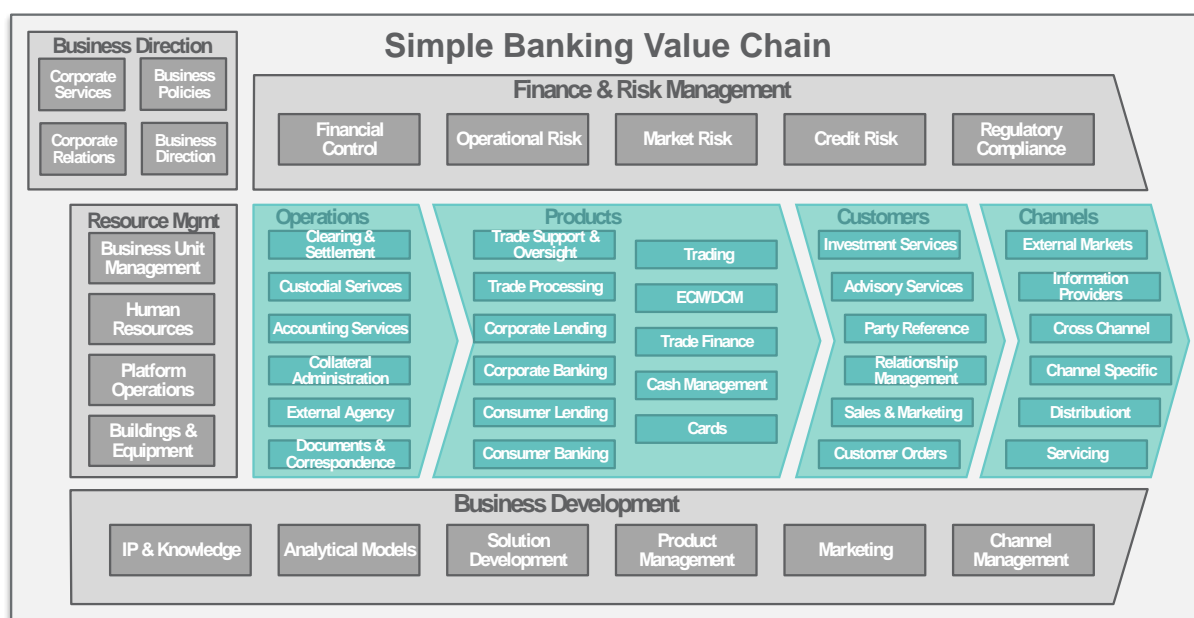


Figure 3: Value Chain Service Landscape Layout

In the landscape the strong candidates for shared services can be seen to surround the product specific services at the center. Banks can determine which of the 300 Service Domains are used in their organization by eliminating product and supporting services that they don't use to create their own version of the "Model Bank enterprise blueprint". This can be a useful tool to communicate the scope of their offerings and activities to their businesses and technology partners.

2.3 – Internal Operation of a Service Domain

As noted the Service Domain handles the full life cycle of all instances of its role being performed. The control record is used to track each instance of the Service Domain performing its role and it reflects a combination of the asset type handled and the functional pattern/generic artifact. Based on practical experience BIAN has found that it is necessary to add a further level of detail to the internal working and governed business information of a Service Domain. This additional detail is needed to ensure the functions performed and the service exchanges are sufficiently defined to be interpreted in a consistent manner from one deployment to another.


This extra detail is defined by breaking down the Service Domain's behavior (as represented by the functional pattern acting on the asset type) into its constituent 'behavior qualifiers'. The way different behavior qualifier types add detail varies greatly for each functional pattern. The standard behavior qualifier types corresponding to each functional pattern are shown in the table:

Functional Pattern	Brief Definition	Information Profile			
		Generic Artifact	Definition/Description	Behavior Qualifier Type	Behavior Qualifier Type Description
DIRECT	Define the strategy	Strategy	The purpose and mission for the enterprise including its competitive positioning and bases for competing in the market	Goals	A collection of goals and objectives for the enterprise and its main divisions
MANAGE	Oversee activity	Management Plan/Charter	The management and oversight while running an operational unit of an enterprise	Duties	A collection of one or more responsibilities or tasks under management
ADMINISTER	Administer activity	Administrative Plan	The clerical support for an operational unit/function of an enterprise	Routines	A collection of one or more clerical routines that are to be followed to administer the operational unit/function
OPERATE	Operate facility	Operating Session/ Facility	The operation of a technical/automated facility employed/provided by an enterprise	Functions	The collection of operational services/functions offered by the operational facility
PROCESS	Process work	Procedure	The performance of a supporting office activity within the enterprise (not product/service fulfillment specific)	Worksteps	The main worksteps to be followed in the execution of the procedure
REGISTER	Register details	Directory	A registry of items recording key reference information and properties required for its use	Properties	The properties/reference details recorded in the registry for items
DESIGN	Design solutions	Specification	A specification of a product or service offering covering all aspects required for its use	Aspects	The main design elements/views making up the overall specification
DEVELOP	Execute projects	Development Project	A discrete or bounded effort with a defined remit and intended purpose/outcome	Deliverables	A collection of one or more deliverables that may be further defined in terms of an approach to be followed to create them
ASSESS	Test compliance	Assessment	A formal evaluation or test of a subject against a predefined set of properties or performance criteria	Tests	A collection of one or more tests/evaluations that can be made to certify a subject
MAINTAIN	Maintain resources	Maintenance Agreement	A service to provide maintenance and repair to operational capabilities/technology	Tasks	A collection of tasks needed to support maintenance and repair work
TRACK	Log events	Log	A mechanism to track and record specific events and if necessary maintain associated derived/accumulated values	Events	A collection of the events/transactions recorded by the log
ANALYSE	Analyse activity	Analysis	A service to apply specific types of analysis against a set of provided data related to an item or activity	Algorithms	A collection of models/calculations/algorithms that can be applied to a subject or activity
MONITOR	Measure resources	Measurement	A mechanism to track and report on the state or dynamic property of some item or activity	Signals	A collection of information feeds/measures that can be used to track the status of one or more items/entities
AGREE TERMS	Govern activity	Agreement	A service to apply specific laws and/or rules to define the terms and conditions that govern a business service or activity	Terms	A collection of terms (within some jurisdiction) that can be selected and configured to define a contract/agreement
ENROLL	Register members	Membership	A registry of entities that qualify for membership to a group with a recognised business purpose or categorization	Clauses	A collection of clauses that govern the eligibility for membership
ALLOCATE	Allocate resources	Allocation	A service to track the availability and allocate business resources (staff and/or facilities) on request	Assignments	A collection of one or more specific assignments of inventory allowing for different allocation types and states
FULFILL	Fulfill agreement	Fulfillment Arrangement	The fulfillment of a financial facility, including customer initiated and internally triggered actions/Features	Features	The product features/services available with a financial facility
TRANSACTION	Execute transactions	Transaction	The execution of a financial transaction	Tasks/Steps	The sub-tasks involved in the execution of the financial transaction

Figure 4: Functional Patterns, Generic Artifacts and Behavior Qualifier Types

The actual behavior qualifiers need to be defined for each Service Domain in accordance with the applicable behavior qualifier type. This additional Service Domain design detail is used in two key ways. The extra definition clarifies the internal behaviors of the Service Domain and so can be used better to define the governed information. The individual qualifiers can also be used when referencing the offered service operations to give a more narrowly defined focus to the purpose of the service operation call.

An example helps clarify how behavior qualifiers add to a Service Domain's definition. Below the Party Authentication Service Domain with the behavior qualifier type 'test' lists the different authentication tests it contains and provides individual services to access each one:



SERVICE DOMAIN	DESCRIPTION	FUNCTIONAL PATTERN	GENERIC ARTIFACT TYPE	ASSET TYPE	CONTROL RECORD	BEHAVIOR QUALIFIER TYPE	BEHAVIOR QUALIFIERS	GENERAL ACCESS OPERATION ACTIONS (EXPANDED)	SERVICE OPERATION
Party Authentication	A cross channel capability that provides contact verification for a customer accessing the bank	Assess	Assessment	Party Authentication	PartyAuthenticationAssessment	Tests	Password Check Secret Questions Document Check Issued Token/Device Check Biometric Match Behavioral Match	Activate: Activate the party authentication facility, Configure: Configure operational parameters, Record: Customer product/service activity and alerts, Evaluate: Combination of any assessment, Evaluate: Customer details/password Evaluate: Customer secret question confirmation, Evaluate: Device identifier checks, Evaluate: Provided document verification, Evaluate: Biometric checks, Evaluate: Behavioral pattern checks, Authorize: NA (authentication not an authorization), Request: Request authentication guidance, Retrieve: Party authentication service reporting,	activatePartyAuthenticationAssessment configurePartyAuthenticationAssessment recordPartyAuthenticationAssessment evaluatePartyAuthenticationAssessmentPassword evaluatePartyAuthenticationAssessmentDevice evaluatePartyAuthenticationAssessmentQuestion evaluatePartyAuthenticationAssessmentDocument evaluatePartyAuthenticationAssessmentBiometric evaluatePartyAuthenticationAssessmentBehavior authorizePartyAuthenticationAssessment requestPartyAuthenticationAssessment retrievePartyAuthenticationAssessment

Figure 5: Service Domain Behavior Qualifiers Added to Service Operations

2.4 – Service Based Access

The next level of the BIAN design outlined here is the definition of the service operation exchanges. A Service Domain provides access to its governed information and functions through offered service operations. BIAN has defined a standard set of 'action terms' that characterize the range of service operation calls:

Action Terms			Description	Example
Origination	Actions to set-up, establish a new control record instance	Initiate	Begin an action including any required initialization tasks	A payment transaction is initiated
		Create	Manufacture and distribute an item	A new analytical model design is created
		Activate	Commence/open an operational or administrative service	The ATM network operation is activated
		Configure	Change the operating parameters for an ongoing service/capability	The on-line ATM's in the network are changed to take machines out of service
Invocation	Actions to access/update/influence an established instance	Update	Change the value of some (control record) properties	A customer's reference details are updated with a change of address
		Register	Record the details of a newly identified entity	A new customer's details are captured
		Record	Capture transaction or event details associated with a life cycle step	An employee logs time spent working on a project against the plan
		Execute	Execute a task or action on an established facility	A payment is applied to a charge card
		Evaluate	Perform a check, trial or evaluation	The eligibility to sell a product is checked against the customer's existing agreement
		Provide	Assign or allocate resources or facilities	A branch requests an allocation of cash for its tellers
		Authorise	Allow the execution of a transaction/activity	Regulatory compliance authorises a product design feature
		Request	Request the provision of some service	A customer requests that a standing order is set up on the current account
		Terminate	Conclude, complete activity	The use of a product version is terminated
Delegation – no new action terms apply as the called Service Domains offer the same Origination/Invocation & Reporting options described here)				
Reporting	Actions to extract details and subscribe to updates	Notify	Provide details against a predefined notification agreement	A unit subscribes to update notifications from the customer agreement service domain
		Retrieve	Return information/report as requested	An account balance is obtained and a report covering activity analysis requested

Figure 6: BIAN Action Terms

A selection of action terms and their associated service operations is most appropriate for a Service Domain depending on its Functional Pattern. Based on this mapping a default set of service operations has been defined for every Service Domain in the BIAN Service Landscape. The default mapping of action terms to Functional Pattern is shown in the table below:

		DIRECT	MANAGE	ADMINISTER	OPERATE	PROCESS	REGISTER	DESIGN	DEVELOP	ASSESS	MAINTAIN	TRACK	ANALYSE	MONITOR	AGREE TERMS	ENROLL	ALLOCATE	FULFILL	TRANSACT
Origination	Initiate																		
	Create																		
	Activate																		
	Configure																		
Invocation	Update																		
	Register																		
	Record																		
	Execute																		
	Evaluate																		
	Provide																		
	Authorise																		
	Request																		
	Terminate																		
Reporting	Notify																		
	Retrieve																		

Figure 7: Default Action Term By Functional Pattern

Where there is a green cell in the table a Service Domain with the corresponding Functional Pattern will offer a default service operation with that action term. The service operation name is a concatenation of the action term and the Service Domain's control record. As an example the default service operations for the Service Domain that handles customer relationships is defined as follows:

Service Domain – Customer Relationship Management

Asset Type – Customer Relationship

Functional Pattern – Manage

Generic Artifact – Management Plan

Control Record – Customer Relationship Management Plan

Behavior Qualifier Type – (Management) Duties

Default Action Terms concatenated with the Control Record defines the default service operations:

activate Customer Relationship Management Plan,

configure Customer Relationship Management Plan,

record Customer Relationship Management Plan,

request Customer Relationship Management Plan,

terminate Customer Relationship Management Plan,

notify Customer Relationship Management Plan,

retrieve Customer Relationship Management Plan

For most of the default service operations the purpose of the offered service operation is unambiguous (refer to the action term descriptions earlier). One where additional specification is often needed is the '*request* Customer Relationship Management Plan' service operation. There are clearly many different types of request that could be made of the relationship management function in different situations.

This is a situation where the behavior qualifiers are used to add the necessary service operation precision. In this case the qualifiers might list the different relationship management duties that would then be used to qualify the service operation (*request* Customer Relationship Management Plan...):

- *Business Development* – matching/proposing products, supporting sales,
- *Budget/Planning* – setting product use and sales goals and refining pricing,
- *Liaison* - Customer interaction, support and trouble-shooting,
- ...Etc..

By linking a behavior qualifier (in this situation the different relationship management *duties*) with the service operation the necessary precision to unambiguously define the purpose of the exchange is obtained. For example:

"request_Customer Relationship_Management Plan_Business Development"

is a more specific service operation that would request relationship management action to address new product and service opportunities. (Underscores added here for clarity only).

2.5 – Defining Message Content – The BIAN API Platform Sandbox

The payload of the service operation is extracted to create the underlying message(s) specifications. The message content will contain or reference an extract of the collection of business information governed by the Service Domain. This extract will usually be a sub-set of one or more control records as determined by the service operation's action term and optionally its behavior qualifier if it has one.

With the latest release BIAN has defined the behavior qualifiers, control record information content and extracted the pertinent information attributes for the service operations for an initial selection of 67 Service Domains referred to as 'BIAN Semantic API Wave 1'. These specifications can be found in the BIAN API Platform Sandbox and are also reflected in the Service Landscape V7.0 release.

BIAN is developing these extended definitions in 'waves' with the first wave covering customer on-boarding, external customer access and control, basic consumer payments and consumer loans. BIAN plans to expand the coverage as fast as practical across the remainder of the service landscape.

An API 'wave' includes examples on the business activities covered using a combination of BIAN Business Scenarios and associated wireframe diagrams. These design artifacts are described in more detail with examples later in this guide. In essence these define the business context within which a service operation is called, i.e. which Service Domain is calling the offered service of another Service Domain for what specific business need.

Using this business context the user/developer can relate the Service Domains and service operation exchanges to the target physical environment. The role/functions performed by Service Domains can be aligned to applications and the service exchanges related to exchanges/interfaces between the applications. The exchanges can be internal to the bank (application to application – A2A) or provide access to external parties either direct to the customer (bank to customer – B2C) or via an intermediary third party (bank to business – B2B).

The service operations provide the high-level definition of an API. BIAN has taken the semantic service operation descriptions and translated the content to a suitable RESTful API format to define a collection of associated endpoint specifications. The precise format and supporting guides and definitions can be found by accessing the BIAN API Platform Sandbox. Access instructions can be found on the BIAN website (BIAN.org)

BIAN is also actively mapping the semantic information definitions for the Service Domains to the ISO20022 Business Model to provide access to more detailed data descriptions that can be reviewed and selections made. With the Wave 1 release some 16 Service Domain control records have been mapped to ISO20022. The mapping can involve making significant extensions and amendments to the ISO model. To manage this BIAN is developing the BIAN Business Object Model which is based on the ISO20022 Business Model but includes the extensions and amendments needed to support the BIAN Service Domain specifications. As explained below, a joint BIAN/ISO team is managing the evolution and coordination of the model with the intent to work towards defining a single integrated view as far as is practical.

3. Key Benefits of using BIAN to design API solutions

The BIAN design approach as outlined is well aligned with API solution development for a number of reasons:

- *Support for Emerging Industry Approaches* – Two key technology approaches are considered: API development and the adoption of a Micro-service architecture
- *Support for Industry Standards* – The BIAN Service Domains and service operations present an Industry standard definition for the componentization and service enablement of Banking
- *Support for Incremental Adoption/Migration* – BIAN aligned solutions can be implemented and adopted incrementally enabling a prioritized migration from constraining legacy architectures

3.1 Support for Emerging Industry Approaches

Two approaches are of particular interest. The first, the subject of this guide is the expanding use of APIs. The second is the growing interest in micro-service architectures.

Application Program Interfaces (API)s – by definition an API supports a software based interface with a software application, most commonly from another software application. An ‘open’ API is a software interface specification that allows an external party to access the bank’s software application from their own software application typically through a managed API environment/platform.

A complete software interface specification needs to consider a wide range of practical implementations such as performance, security, technical environment/architecture in addition to the more obvious function/logic and data/information specifications.

To implement a truly interchangeable software interface comprehensive specifications must be defined and applied. For anything more than very basic exchanges these specifications will be very complex. Given that these interfaces typically need to relate to existing applications, these specifications will often also include a high degree of proprietary or site specific detail.

The banking industry is making progress defining standard specifications for some more common transactional exchanges, in the area of payments for example. But even in these areas where there is a long history of standards based message interfaces it is proving difficult to define precise industry specifications that do not require site-specific interpretation and some degree of host software mapping/reworking.

The value that the BIAN model provides with the development of open APIs is by providing an approach by which this specification challenge can be managed and its implications minimized. By defining a business exchange in terms that can be consistently interpreted the BIAN Service Domains and service operations help in two specific ways:

1. The service interaction between two Service Domains can be defined in sufficient detail that when used as a high level specification for an API the business purpose and key business information exchanged is standardized. Two software implementations mapped to the same service interaction will typically differ in their finer implementation details but if an external user of the service chose to switch between the two suppliers the disruption would be contained to the software in the immediate vicinity of the exchange itself. It should be possible to map to the key business information content to the changed data schema. More importantly the main function/logic should be consistent so that any up-stream/down-stream activities at the caller are not significantly disrupted.
2. Because the Service Domain supports a narrowly defined business function and the service operations invoke specific responses the associated messages have highly focused business information content. As a result the extent of the business information to be mapped (and the associated meaning/purpose of the information) is kept to a practical minimum.



Micro-services – micro-service architecture has a lot in common with the core design principles employed by BIAN. The Gartner definition of a Micro-service underscores this:

“A micro-service is a tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable application component.” – Gartner

Micro-services can be defined at varying levels of detail, indeed the terms ‘*nano service*’ and ‘*macro service*’ are often used to describe finer and coarser grained components respectively. At one level the boundary of a Micro-service can be mapped directly to the role of a Service Domain. The functioning of the Service Domain is then the same as the associated micro-service component and the offered and consumed Service Domain service operations define the Micro-service boundary.

Because a Service Domain performs a single discrete function and in particular because it handles all instances of its specified business role from start to finish the Service Domain has very strong function and data partitioning. Furthermore when a Service Domain is implemented following proper service oriented design the service behaviors can strictly enforce encapsulation.

It can be argued that the BIAN partitioning approach defines business components that specifically conform to the goals of micro-service design. The summary table below outlines how BIAN Service Domains and Micro-services can be compared:

Level	Services	Application Integration
<i>The hierarchy used to build up solutions from elementary components</i>	<i>Defines business capability partitions as discrete and bounded (static) functionality</i>	<i>Defines the exchange of information and actions to support (dynamic) business behaviors</i>
Elemental Component	 <p>The BIAN Service Domain provides a candidate conceptual/logical design for a Microservice. (Note: There may be multiple physical interpretations of the Microservice design in physical implementation)</p>	 <p>A BIAN service exchange between two Service Domains is an elemental interaction defined in terms of context, purpose & information payload. (Note: This is a conceptual/logical specification that can be applied to defining and implementing an API)</p>
Bundles – initial thinking is that bundles need to be support at two levels: <ol style="list-style-type: none"> 1. Business Applications 2. Organizational Entities 	Service domain 'clusters' can be described using BIAN wireframes, business scenarios and more detailed Service Domain & service operation specializations	"API Solution Sets" can combine collections of 'elemental' exchanges as may be required to define internal and external exchanges within and between business applications within an enterprise. Also to provide external access to that enterprise ("Open APIs")

"A microservice is a tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable application component."
– Gartner

Figure 8: BIAN Service Domains Related to Micro-services

Furthermore, because BIAN Service Domains handle all instances of their control records for the complete life cycle they 'encapsulate' their own business information. Just as the role of each Service Domain is discrete so is the profile of the business information that it governs. The Service Domains collectively cover all of the Banks activity so the combination of the information they govern and they exchange through service operations defines an information architecture that shows where all business information is managed and how it flows through the organization.

The effective partitioning and encapsulation of business information is a key enabler for a micro-service and indeed any highly distributed architecture. The BIAN Service Domains define autonomous business partitions that are particularly well suited to this type of architectural design.

3.2 Support for Industry Standards

BIAN as an Industry Standards body defines a unique business architecture model of banking activity that outlines the discrete, canonical functional components and service exchanges as described in this guide and other BIAN documents. There are two other industry standard with which BIAN is maintaining close alignment that are of specific interest to API development:

1. *ISO20022*. The BIAN Metamodel is based on the ISO messaging standard (S.W.I.F.T., the 'custodian' of the ISO20022 model is a founding member of

BIAN). BIAN continues to work closely with the relevant ISO working groups to ensure that the standards remain aligned and that any content BIAN develops builds on ISO content and similarly that any new content BIAN develops is provided for consideration by the ISO operation when appropriate.

BIAN has adopted the ISO20022 Business Model as the foundation for defining its business object model – the BIAN BOM. In order to fully relate the content of the business object model to the BIAN Service Domains it has proven to be necessary to extend and enhance the ISO20022 model significantly. The BIAN BOM combines the ISO20022 Business Model with these extensions and amendments. The hope and intent is that BIAN and ISO can coordinate efforts and work towards a single integrated object model.

As BIAN develops the extended content for the API initiative a specialist team within BIAN that includes ISO specialists from S.W.I.F.T. works to maintain the alignment and define extensions to the ISO20022 model. In particular these extensions address the required link between the Service Domains, their control records and the corresponding objects in the ISO model

The BIAN team documents any potential additions to the ISO model and registers it for consideration in future versions of ISO20022.

2. *OMG & EDM Council - FIBO* – a joint effort between the EDM Council and the Object Management Group, the Financial Industry Business Ontology (FIBO) can be thought of from one perspective as an industry thesaurus. It defines banking concepts and allows for these definitions to be specific to a particular business context, maintaining synonyms and homonyms in addition to the conceptual definitions. As BIAN builds out its own BOM, reference is made to the content of FIBO and the terms used where possible. The content development of FIBO is at an early stage and so the precise mechanics of this collaboration is likely to evolve.

3.3 Support for Incremental Adoption/Migration

Perhaps one of the more important reasons for aligning to the BIAN standard when developing APIs is a key property of the BIAN Service Domains' partitioning. This property is that the BIAN Service Domains define aspects of banking that are ***extremely stable over time***. The role or purpose of a Service Domain does not change as business practices evolve.

This is because the Service Domain represents an elemental responsibility or something that the bank needs to be able to do. It does not prescribe how that particular responsibility is fulfilled. For example one BIAN Service Domain is responsible for providing customer authentication services to confirm the identity of a customer when they present themselves to the bank for any reason.

As noted earlier this is a “canonical” requirement in that it is a function that every bank can agree on the need for. In practice different banks may use different methods/techniques in different combinations to actually verify a customer (for example passwords, device identifiers, biometrics) and indeed over time the possible mechanisms may evolve as new approaches are invented. But all banks can agree that they need to be able to verify a customer’s identity.

The value of the BIAN model here is that it defines the persistent functional partition and its role/purpose in a way that can be consistently referenced. In addition the BIAN model defines the different services that are likely to be needed to access the facility and provides a standard ‘semantic’ description of the information exchanged. These specifications are intended to reflect prevailing ‘mainstream’ behaviors for the Service Domain.

Though the purpose and boundary defined by a Service Domain is stable, the way the Service Domain executes its role internally can vary and will most likely evolve with new practices and innovation. Also the patterns of interaction with other Service Domains in response to different business events and the thresholds triggering service operation exchanges may vary from site to site and over time. But as noted the fundamental role of the Service Domain and where it fits amongst the other Service Domains should not change.

As a result, APIs that are aligned to BIAN Service Domain service operation boundaries can be designed for incremental adoption. Some basic services can be supported and adopted in limited situations initially and over time additional services either supporting more complex interactions with existing Service Domains or adding new Service Domains into the mix can be supported without destabilizing existing capabilities.

By selecting high-value, low-complexity API services to start, business benefits from early solutions can often help fund the overall migration and build momentum for adoption by proving positive business impact as new services are offered over time.

4. Using BIAN as an API 'Inventory'.

The BIAN Service Landscape is a reference structure that contains the BIAN Service Domains in a layout intended to help with their identification/selection. BIAN seeks to identify all required Service Domains to support any and all banking activity in the Financial Services Industry. Furthermore by defining the service operations offered by each Service Domain the BIAN Service Landscape represents a complete inventory of service interactions at a particular level of detail.

For this reason the BIAN Service Landscape can be used as an organizing framework for categorizing available APIs. By mapping an API to the corresponding service operation(s) for the providing Service Domain it can be uniquely classified. As the inventory is populated with references to available open APIs users will be able to identify potential solutions for specific purposes. As noted earlier there is always likely to be implementation and mapping work to do to deal with practical aspects of the API implementation, but the addressed business requirement can be well matched.

The scope of an available API solution may map to more than one BIAN service operation or may provide a subset/more specialized service. Because the Service Domains serve 'elemental' business purposes and also their service operations are narrowly defined the mapping should usually be fairly straightforward.

With the launch of the BIAN API Platform Sandbox BIAN has provided candidate RESTful API specifications for 67 selected Service Domains (resulting in some 900+ endpoints). BIAN intends to expand coverage across the remainder of the landscape as quickly as practical. In anticipation of these extended specifications and as already noted the existing higher-level BIAN Service Domains and service operations can be used to classify/categorize APIs in the interim.

In order to develop the API inventory a review of the BIAN Service Landscape was made to make an initial determination as to which Service Domains provide function and business information that might sensibly be accessed externally i.e. for B2C (bank to customer) or B2B (bank via intermediary/3d party – bank to business) interactions. The Service Landscape below has been color coded to show this classification. Service Domains highlighted in green represent business functions that provide cross product or utility type services. Service Domains highlighted in red represent business functions that are specific to a particular product. Note: This is only an initial determination for planning purposes. It is highly likely that other patterns of business behavior will be discovered that may make additional Service Domains of interest.

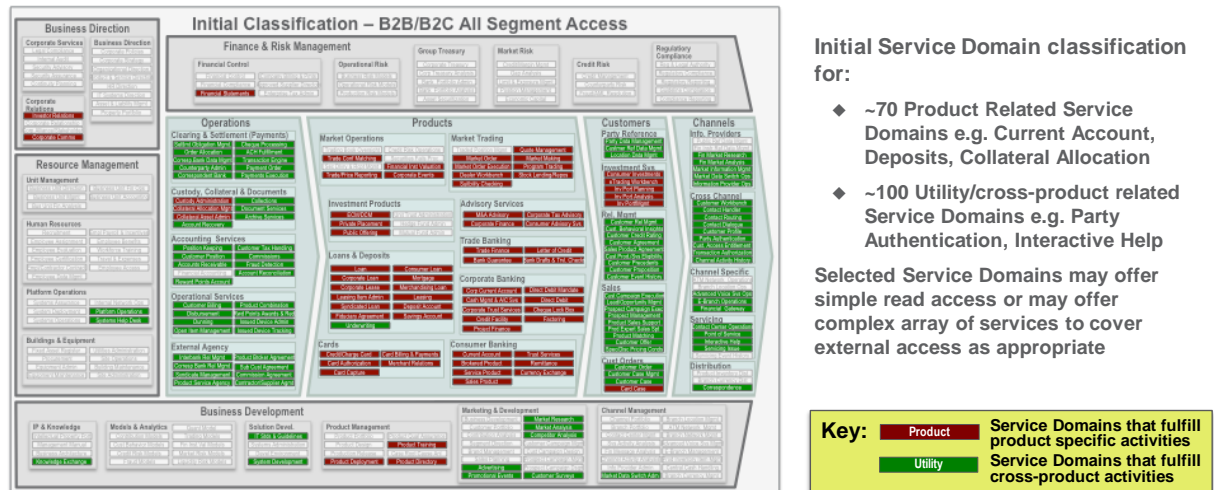


Figure 9: The Service Landscape with Open API Candidates

Every Service Domain has an associated set of default service operations. Some of these support internal management command and control type exchanges and these are unlikely to be offered through open APIs. For this reason they have not been included in the API specification at this stage. For all other service operations a brief description has been provided so that open APIs can be mapped against the associated Service Domain and service operation(s).

These service operation descriptions are being refined as BIAN develops extended definitions of the Service Domains. The coverage of the first phase of this design extension is presented later in this guide in the section covering 'Wave 1' deliverables. An example of the outline descriptions of the service operations is shown in the table:

Business Direction

Current Service

Business Direction

Initial Classification – B2B/B2C All Segment Access

Finance & Risk Management

Operations

Products

Customer Relationship Management

Channels & Distribution

Business Management

Human Resources

Marketing & Sales

Business Development

IT & Technology

Business & Analytics

Key:

Product

Utility

Service Domains that fulfill product specific activities

Service Domains that fulfill cross-product activities

Current Account

Default BIAN Service Operations	Service Description/Objective
initiateCurrentAccountFulfillmentArrangement	Initiate/set-up a new current account facility
updateCurrentAccountFulfillmentArrangement	Update reference details of an active current account arrangement
recordCurrentAccountFulfillmentArrangement	Record activity/feedback against a current account arrangement
executeCurrentAccountFulfillmentArrangementDeposit	Execute a deposit transaction against the account
executeCurrentAccountFulfillmentArrangementPayment	Execute and payment transaction against the account
requestCurrentAccountFulfillmentArrangementStandingOrder	Request the set-up/amendment of a standing order
requestCurrentAccountFulfillmentArrangementDirectDebit	Request the set-up/amendment of a direct debit
requestCurrentAccountFulfillmentArrangementSweep	Request the set-up/amendment of an account sweep arrangement
requestCurrentAccountFulfillmentArrangementInventory	Request the issuance of product inventory (cheque book, plastic, etc)
requestCurrentAccountFulfillmentArrangementClose	Termination request for an in force current account
retrieveCurrentAccountFulfillmentArrangement	Retrieve current account reports (e.g. balances, statements, account configuration)

Business Scenario (Number & Name)	Service Exchange - Calling Service Domain and Goal	Action/Sub-Action	Offering Service Domain	Pre-conditions	Request Parameters	Response Parameters	Post-conditions						
					Identifiers	Depiction	Instructors	Analyses	Identifiers	Depiction	Instructors	Analyses	
13 - TPP obtains current account statement	TPP/Client - Contact Dialogue. The TPP passes on the user request to check their balance	Execute/Balance Query	Bank - Current Account	Active contact - user is connected with the TPP (via a concurrent session) and provides their requests that are passed on by the TPP/Bank session modelled here	User reference, Current account reference	NA	Execute/Balance Query	NA	User reference, Account reference	Account balance	NA	NA	Account balance provided

Figure 10: Example Wave 1 – Service Operation Descriptions

The identification of 'open' APIs addresses external B2B and B2C access to the bank. But in order to consider how a bank handles the fulfillment of a service request internally it is usually necessary to trace the 'downstream' A2A interactions as well. These internal interactions can be used to specify internal APIs that can be used to rationalize internal application architectures – indeed supporting the rationalization of internal applications is the founding mission for BIAN. For this reason the extended BIAN specifications cover all transactional service exchanges whether they be B2B/C or A2A. The use of A2A APIs is considered in more detail in the next section where different API alignment approaches are considered.

5. Three types of architectural alignment

The implementation of open APIs varies greatly depending on the technical architectural approach adopted. BIAN defines three distinct types of technical solution. There may be hybrid architectural approaches and in practice most banks will need to consider a range of technical situations that might span all three types for different aspects of their operation.

1. **Direct to Core** – in this approach basic external access control is handled in the channel (or via some dedicated front-end control mechanism) and the service exchange accesses the host facility directly.
2. **Wrapped Host** – in this approach access control remains channel based but the host systems are accessed through a control/wrapping middleware such as an enterprise service bus (ESB)
3. **Distributed Architecture** – in this approach the applications are implemented as a network of service enabled, discrete capabilities that can support external access directly

Key properties of each and their business rational is summarized in the table:

	Type 1. Direct to Core	Type 2. Wrapped Host	Type 3. Distributed Architecture
Definition	The API routes direct to the core system providing the service. Intermediate channel based access control and 'buffering' is required	Integrating service middleware – a service bus – 'wraps' the host systems. The service bus can offer various host access mitigation capabilities/enhancements	The host services can be implemented as loose coupled 'micro-services' with complex interactions supported by sophisticated connective middleware
API Service Description	Read only or simple 'atomic' update transactions supported by a single host system. The solution is likely to be host application specific	Enhanced 'simple access' services aligned to established standards. Wrapping may enhance service capabilities and some hosts may support more complex exchanges	Support for flexible and complex interactions involving multiple business activities and processing/decision chains
Examples	<ul style="list-style-type: none"> ◆ Retrieve a balance/account statement ◆ Reference a product/service directory ◆ Initiate a payment 	Message conforms to industry standards (e.g. ISO20022) <ul style="list-style-type: none"> ◆ Retrieve a balance/account statement ◆ Reference a product/service directory ◆ Initiate a payment ◆ Customer on-boarding/offers 	<ul style="list-style-type: none"> ◆ Prospect on-boarding and origination ◆ Customer dispute/case resolution ◆ Customer relationship development/up-sell/cross-sell campaigns ◆ Third party service integration
Business Drivers	Provide application based access to an established/existing type of customer exchange	Provide application based access with a high degree of standards alignment. Mask/augment host/legacy system limitations.	<ul style="list-style-type: none"> ◆ Support sophisticated interactions ◆ Support new business models ◆ Support for 3rd party integration ◆ Leverage advanced technologies/architectures

BIAN
Aligned

Figure 11: Summary of API Sophistication Levels

5.1 Direct to Core

The first type and the easiest to implement involves constructing a front-end capability to manage external access security and then typically re-packaging existing host interfaces to support an API. A typical arrangement is as shown that shows direct customer access to the Bank (from an API linked to their personal device) or via a third party service provider:

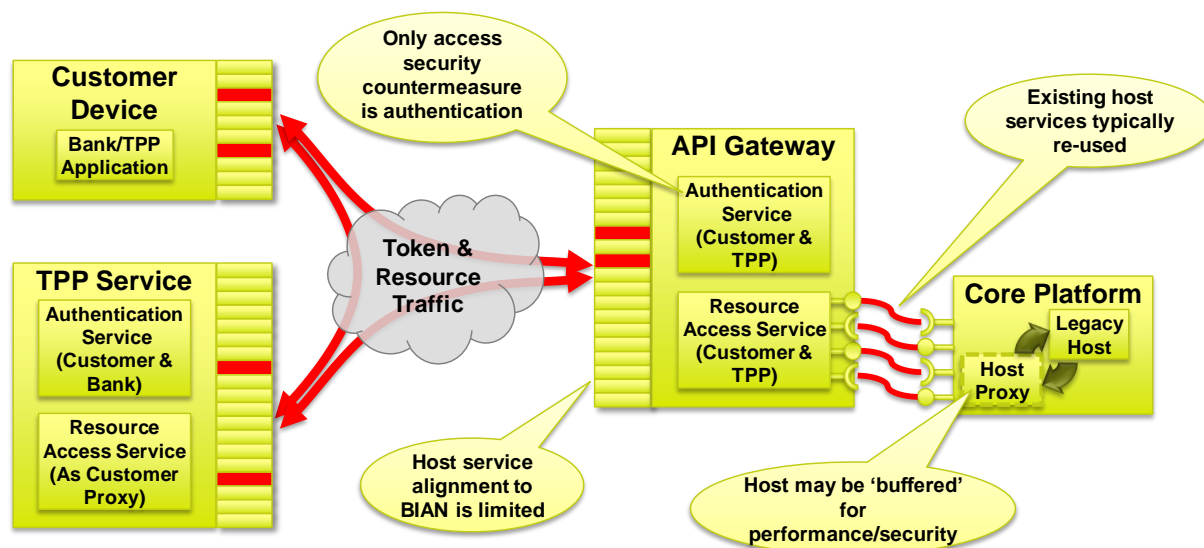


Figure 12: Type 1 Layout

Key Aspects of the Approach

With this approach the changes required of host systems are kept to a minimum but the facilities that can be supported are limited to repackaging existing services that can be accessed through an API front-end platform.

External access control is implemented using access tokens handled by an authentication service capability. Access sessions will typically be limited to single task exchanges that target an individual host system.

Host access may be direct or host production systems may have a front-end 'proxy' implementation that duplicates aspects of the host system to provide additional access control/security/performance.

API services can be mapped/classified against BIAN Service Domain service operations. It is likely however that there will be significant host system specific features exposed through the API.

Advantages

- **Minimal reworking of core production systems** – existing external access interfaces can be repurposed to support API access. Solutions can be developed quickly with limited disruption
- **Access Control** – robust authentication services can be employed to manage external access control independently augmenting and with limited disruption to existing production capabilities

- **Basic Standards Alignment** – API services can be packaged/aligned with the BIAN standard to ease future migration to more advanced solution architectures. (System/software changes will be needed, but the business purpose of the service will be consistent)

Limitations

- **Simple Transactions Only** – the supported services will be simple – read only and/or limited transactional initiation services supported by a single host application access session.
- **Limited/No Post Access Transaction Tracking** - supporting the external tracking of downstream progress of initiated transactions will not usually be possible without significant additional development
- **Authentication Only Access Control** – more sophisticated security countermeasures such as behavioral analysis and cross-contact invocation limits/constraints can't be supported

5.2 Wrapped Host

The second type of approach involves the integration of a host access middleware layer that mitigates host systems shortfalls. The middleware, typically some form of enterprise service bus (ESB) can provide a range of enabling facilities including:

- *Host Access Session Management* – supporting host access ‘sessions’ that can span multiple external access events
- *Data Caching* – persisting frequently accessed host data to minimize host access traffic
- *Host Wrapping* – adding function and data to mask host system shortfalls
- *Resolve Data Fragmentation* – enforcing master/slave data governance techniques within the application portfolio
- *Advanced Look-up* – using access patterns to anticipate needs and obtain host data in advance to minimize host access latency
- *Transaction Persistence* – provide facilities to track customer ‘transactions’ between contacts and potentially transactions spanning multiple systems

Again customer access can be direct or via a third party service provider and front-end authentication remains the main security countermeasure.

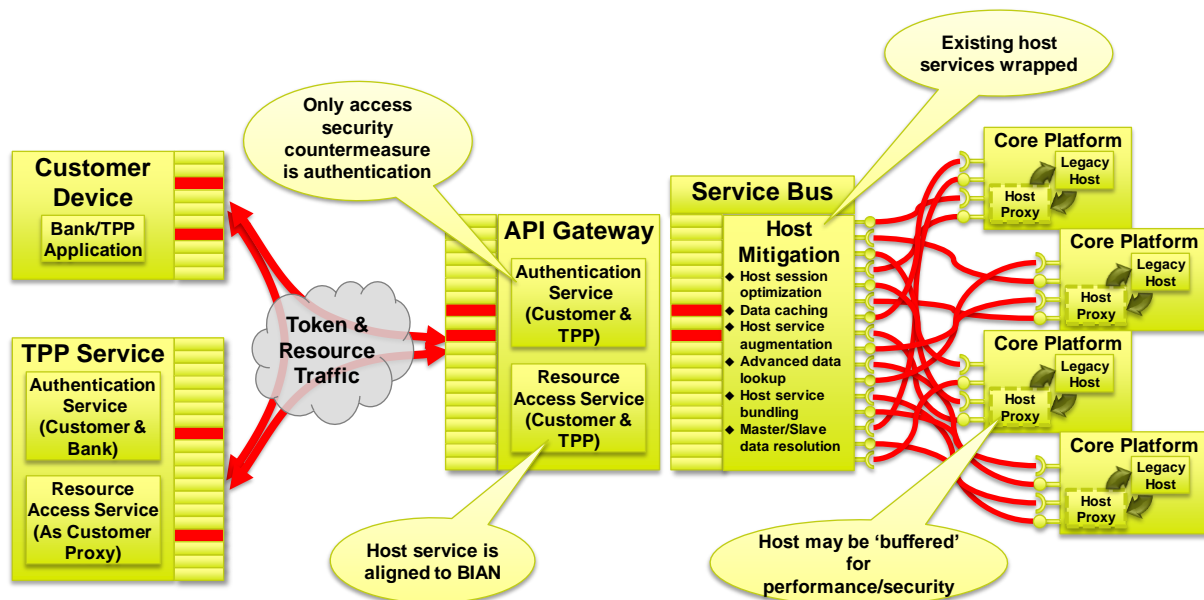


Figure 13: Type 2 Layout

Key Aspects of the Approach

The main purpose of implementing a host-wrapping layer is to repurpose or extend the life of existing legacy systems and enable greater re-use of business functionality. In addition to addressing the listed shortfalls and improvements API services are mapped/classified to BIAN and the ESB wrapper can be used to mask host specific features improving the standards alignment.

Wrapped host services can also support front-end (client-side) application assembly approaches but this type of solution development is not shown in the diagram or considered here in any detail.

Advantages

- **Core Renewal** – core/legacy systems can be repurposed to extend their production life, leveraging sunk investment and avoiding production disruption
- **Access Management** – the ESB can optimize host access reducing systems loading and ensuring unintended patterns of host access are not permitted
- **Full Standards Alignment** – the ESB can mask host specific features allowing services to conform to open standards as far as they are available
- **Support for Front-end Application Assembly** – as already noted

Limitations

- **Limited Complex Transaction Support** – the ESB may be able to support customer transactions that involve some level of cross contact session persistence and/or multiple system access.
- **Limited Front-end Application Support** – the ESB environment is typically not itself intended as the platform for assembling elaborate front-end application capabilities – additional facilities are usually required
- **Authentication only Access Control** – as with type 1 solutions, the wrapped host architecture will use front-end access control mechanisms limited to authentication based countermeasures

5.3 Distributed Architecture

The most sophisticated type of approach is where the host systems conform to a micro-services architecture (or some technical equivalent) with Service Domains (or groups of closely related Service Domains) acting as autonomous service ‘containers’ in a loose coupled service network. In this configuration a particular collection of Service Domains manages customer access, providing comprehensive services including access security, activity tracking and intelligent routing decisioning.

The front-end customer access platform that manages external access can link to different host configurations. The diagram below shows how a customer access platform allows managed access to host systems conforming to different types of API sophistication (Direct to Core, Wrapped Host and Distributed/Micro-service configurations).

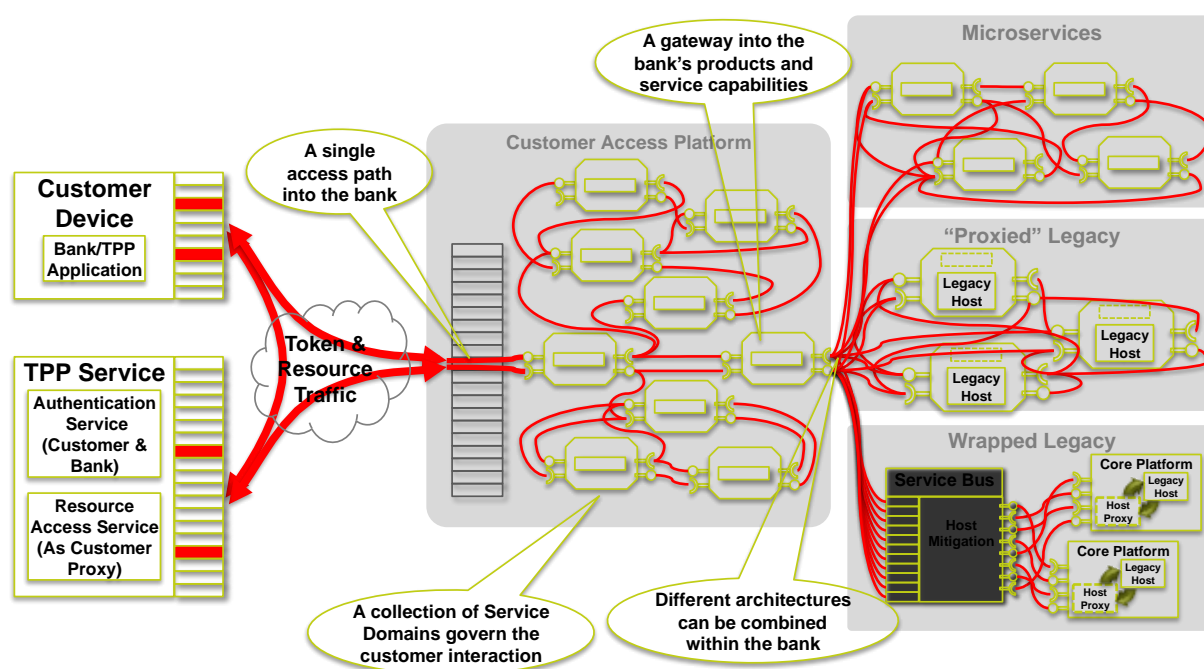


Figure 14: Type 3 Layout

Key Aspects of the Approach

The distributed architecture approach needs to be considered in terms of two distinct aspects. The first as mentioned is the customer access platform that may include a range of facilities and utilities that support external customer access again possibly through third party intermediaries. The second is the bank's product and service capabilities that may increasingly be supported using systems conforming to a distributed/container based architecture where this is appropriate.

As noted earlier, a key advantage of aligning to the BIAN Service Domain and service operation standard for type 1 & 2 solutions is that these interfaces can be later integrated with a type 3 front-end access platform with manageable amounts of re-working.

Advantages

- **Support for Complex Transactions** – the customer access platform can include capabilities to orchestrate complex transactions that span contact and integrate multiple host capabilities over time
- **Comprehensive Access Security Countermeasures** – the customer access platform can integrate behavioral tracking capabilities and other security countermeasures in addition to authentication services

- **Support for Flexible Business Models** – a distributed/container based architecture better supports integrating third party solution elements to support enhanced and completely new business models
- **Enables Background Migration of Core Systems** – a customer access platform can help manage the progressive evolution of legacy systems (through type 1 to type 3 application migrations)
- **Supports Integration of Advanced Solutions** – a distributed architecture can better integrate components/container based designs that leverage advanced technologies for their own internal implementation

Limitations

- **Represents a Significant Investment** – the development of a customer access platform and distributed/container based solutions will involve significant development costs and employs advanced techniques with elevated implementation risk that should be approached incrementally
- **Unproven in Production/Unpredictable Business Case** – leveraging highly distributed/componentized architectures to support new business models can result in unpredictable business results/practices
- **Evolving Standards (BIAN/ISO/FIBO)** – comprehensive development standards covering all aspects of contained based design have not been defined and are emerging at varying pace. Early development risks non-conformity with later standards

6. BIAN Design elements used in API specification.

BIAN is developing extended Service Domain and service operation specifications across the Service Landscape. The various design artifacts that are used to support API design are described here. Later in the guide specific examples from Wave 1 of the BIAN Semantic API initiative are presented – these represent the first installment of the extended definitions that are included in the SL V7.0 release and that have been translated into RESTful APIs in the BIAN API Platform Sandbox.

- *Extended Service Domain Specifications* – an additional level of design specification has been added to the Service Domains to ensure consistent interpretation of the business purpose behind the service operations
- *Wireframes (showing Enterprise Boundaries)* – wireframes present the collection of Service Domains and their service operation connections that support some aspect of business operation. In Wave 1 the wireframes are adapted to show external (3rdparty) activity alongside internal bank flows
- *Enhanced Business Scenarios* – the BIAN business scenario definition has enhancements to clarify the reference to specific business information exchanges (service operation connections)
- *Service Operations (End Points)* – Individual service connections are described in more detail to support their adoption in API design for both external (B2B/C) and internal (A2A) traffic. As noted these service operation definitions have been translated into RESTful API specifications in the BIAN API Platform Sandbox

6.1 Extended Service Domain Specifications

The key additional design concept employed to extend the Service Domain and service operation specifications is the ‘behavior qualifier type’. This provides a breakdown of a Service Domain’s behavior as represented by its functional pattern as described earlier in this guide. The behavior qualifiers defined for a Service Domain are used in two main ways. One, they are used to provide a more detailed definition of the business information governed by a Service Domain (which feeds into the message content for its offered services). Two, they are used to provide greater precision to the purpose of the offered service operations. The extended definitions for some Service Domains taken from the Wave 1 content is shown in the following Excel extract:

SERVICE DOMAIN	DESCRIPTION	BEHAVIOR QUALIFIERS	DEFAULT ACTION TERMS	GENERAL ACCESS OPERATION ACTIONS (EXPANDED)	SERVICE OPERATIONS
Party Authentication	A cross channel capability that provides contact verification for a customer accessing the bank	Password Check Secret Questions Document Check Issued Token/Device Check Biometric Match Behavioral Match	Activate Configure Record Evaluate Authorise Request Retrieve	Activate: Activate the party authentication facility, Configure: Configure operational parameters, Record: Customer product/service activity and alerts, Evaluate: Combination of any assessment, Evaluate: Customer details/password Evaluate: Customer secret question confirmation, Evaluate: Device identifier checks, Evaluate: Provided document verification, Evaluate: Biometric checks, Evaluate: Behavioral pattern checks, Authorize: NA (authentication not an authorization), Request: Request authentication guidance, Retrieve: Party authentication service reporting,	activatePartyAuthenticationAssessment configurePartyAuthenticationAssessment recordPartyAuthenticationAssessment evaluatePartyAuthenticationAssessmentPassword evaluatePartyAuthenticationAssessmentQuestion evaluatePartyAuthenticationAssessmentDevice evaluatePartyAuthenticationAssessmentDocument evaluatePartyAuthenticationAssessmentBiometric evaluatePartyAuthenticationAssessmentBehavior authorizePartyAuthenticationAssessment requestPartyAuthenticationAssessment retrievePartyAuthenticationAssessment

Figure 15: Expanded Service Domains (Excel)

In the example table the behavior qualifiers can be seen mapped to the default action terms. Also shown is the way the behavior qualifier can be used to define a more precise service operation by using it as qualifier component of the service operation name – the different authentication tests defined by the qualifiers are used to stipulate different ‘evaluate’ service operations. The result is the breakdown of the single action term service operation into a collection of finer-grained service operations.

6.2 Wireframes (showing enterprise boundaries)

The BIAN wireframe is an informal artifact that shows how a suitable selection of Service Domains with their associated service connections supports some aspect of the business. A wireframe is a static ‘map’ showing the required service operation connections between the selected Service Domains. The flow of service operation connects that would result from a business event can be traced as a ‘dynamic journey’ across the static map of the wireframe. It is a useful framework view for overlaying current and planned physical systems to show gaps, overlaps and misalignments.

For the Semantic API content the wireframe view has been adapted to show the structures within and between entities that may interact with the bank, including the customer, third party solution providers, network providers and regulators. This view helps isolate where there are external bank to business (B2B) and bank to customer (B2C) interactions that are likely to be supported by open APIs. The internal application to application (A2A) connections are used to trace the required internal interactions for the resolution/down-stream processing of the initiated tasks/transactions.

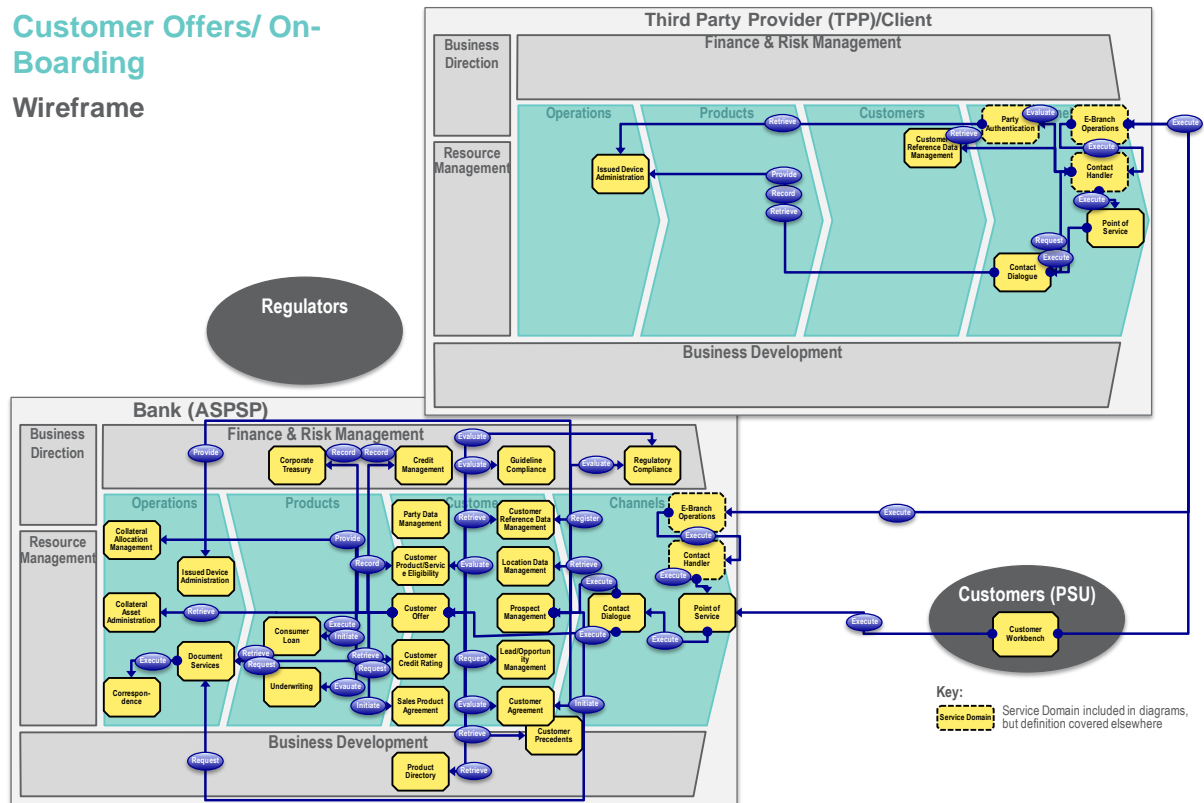


Figure 16: Wireframe Example from Wave 1

In the example wireframe the applicable service operation action term is shown on the arrows linking the Service Domains. The arrow points from the offering Service Domain to the calling Service Domain. As can be seen for the Wave 1 layout the Service Domains have been overlain on the Value Chain structure of the BIAN Service Landscape to help clarify the types of interactions involved.

A further classification of the Service Domains can be considered, showing the time dependency between Service Domains for service operation exchanges. This will usually be an implementation specific property. It can be useful to indicate the likely configuration as a starting point to clarify where service operation exchanges may have critical start/end timing dependencies. An example classification of these dependencies is shown in the mobile access wireframe below:

Mobile Access

The customer accesses the bank (via the 3rd party), is authenticated and goes through the exchanges needed to capture and confirm the 'order'

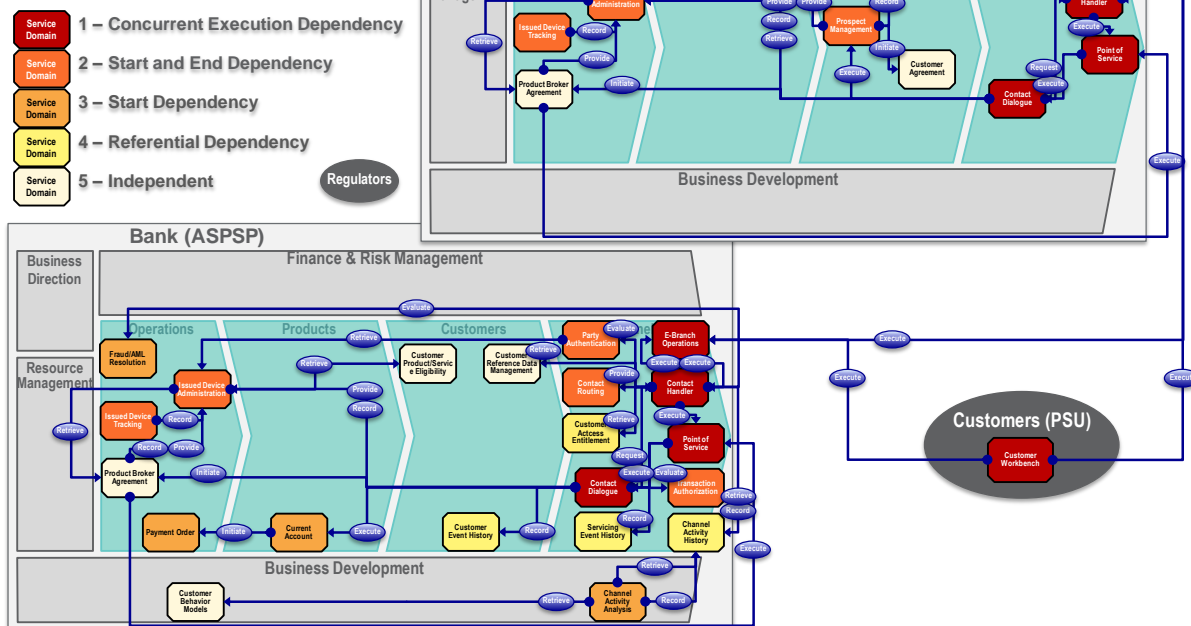


Figure 17: Mobile Access Wireframe with Time Dependencies

6.3 Business Scenarios

BIAN Business Scenarios are another informal BIAN design artifact, depicting an archetypal flow of the Service Domain service connections for a defined business event. The scenario is not intended to be prescriptive nor comprehensive but simply shows a feasible flow between the involved Service Domains of interest clarifying the purpose of the service connections by example. For the Semantic API specification the normal layout has been extended to show the boundary between the bank and other interested parties (a vertical black line delimits Service Domains within each operating entity).

The scenario template also shows the key business information exchanges between the source (calling) and consuming (offering) Service Domains at the bottom of the diagram. These exchanges are tagged to the offering service operation of the called Service Domain. This matched service operation provides the description of the semantic business information content that would need to be exchanged through an API.

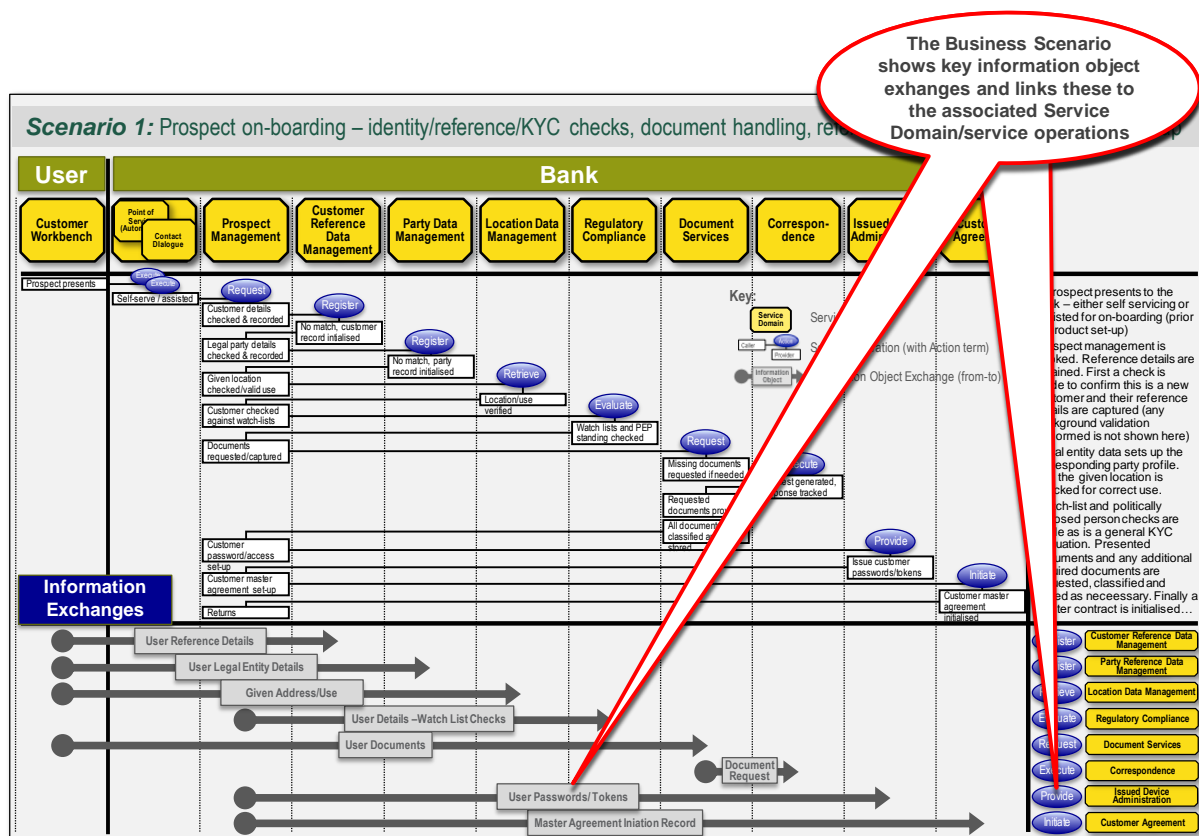


Figure 18: Example Business Scenario

The combination of a wireframe and the associated collection of example business scenarios provides an overview of the business context for service connections that align to B2B/C and A2A interfaces between systems when they are mapped to a bank's own application portfolio.

6.4 Service Operations

The final design artifact of interest for API design is the service operation description itself that underlies the wireframe and business scenario examples. With the Wave 1 content BIAN has translated the extended Service Domain and service operation definitions for an initial selection of 67 Service Domains into a suitable pseudo code form to provide a starting point for API solution development. As already noted these API specifications can be reviewed in the BIAN API Platform Sandbox where the default service operations of the selected Service Domains have generated in excess of 900 RESTful endpoints.

As the extended definitions of the Service Domains and service operations are produced for selected business activities (with the associated wireframes and business scenarios) BIAN lists indicative semantic information content based on the business information profile of the Service Domain as captured in its control record. As already described, the business information profile combines properties of the Service Domain's asset type and its generic artifact as represented by the control record.

For the extended definitions (supporting the API initiative) the control record content has been defined by listing all of its key information attributes. These 'candidate' or initial attribute lists will be refined based on feedback from members and others that reference the BIAN API specifications.

As noted BIAN is also developing a comprehensive business object model, building on the ISO20022 Business Model. The Service Domain control record attribute lists are mapped against the BIAN BOM to provide standard definitions and to link into more detailed underlying ISO specifications where available. The mapping exercise requires extensions to the ISO model to support the required linkage to the BIAN Service Domain and control record structures. This mapping has been completed for 16 of the 67 Service Domains included in Wave 1. The Service Domains most likely to support B2B/C exchanges were given priority.

As BIAN continues to develop extended Service Domain specifications across the remainder of the Service Landscape a coordinated effort to extend the BIAN BOM and map the content to the service operations will run in parallel. In time the BIAN service operation specifications (and the underlying RESTful API specifications) will reference ISO definitions where they are available with the ISO attribute definitions linked to or replacing the control record attribute lists as appropriate

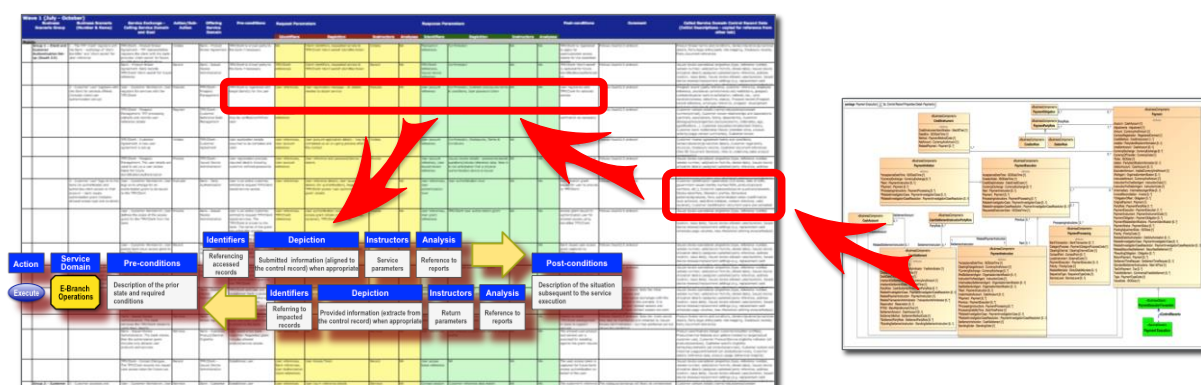


Figure 19: Service Operation Definition

7. BIAN API Design Approach

In practice banks may need to deploy solutions at all of the three types described in this section. The advantage of alignment to the BIAN API specification is that it is easier to combine solutions implemented using the different types of architecture and to migrate solutions from one type to another when necessary. The descriptions outline some general considerations and justifications for adopting the different approaches. These guidelines are intended for architects developing the overarching approach for API adoption. More detailed implementation/practitioner guidelines for RESTful endpoints are being integrated into the BIAN API Platform Sandbox and its supporting documentation.

An outline approach for applying the BIAN semantic API designs is set out under the following sub headings:

- *General Implementation Considerations* – an earlier BIAN API guide listed a number of techniques that could apply in different implementation situations when using the BIAN service connections as a high level API design element. These are repeated here for reference. In addition a Service Domain grouping technique BIAN uses called clustering is outlined
- *For Type 1 Direct to Core* – describes tasks specific to aligning API design work for level 1 type implementations
- *For Type 2 Wrapped Host* – describes tasks specific to a level 2 type implementation i.e. using some form of wrapped host middleware or enterprise service bus (ESB)
- *For Type 3 Distributed architecture* – describes tasks that help establish both a customer access platform and the broader implementation of micro-service based products and services

7.1 General Implementation Considerations

An earlier BIAN API guide went into some detail defining different techniques that could apply with the implementation of an API aligned to a service connection between two Service Domains. As applicable techniques are refined and additional associated industry standards are discovered these more detailed techniques will be updated and made available as appropriate in future releases.

The selection from the applicable techniques provides insight to a number of practical considerations for implementing a suitable API design. The BIAN specification provides a description of the business event/purpose for an individual service operation interaction between two Service Domains and also provides a semantic description of the exchanged business information. The BIAN specification does not attempt to address any of the more detailed and often largely site-specific software implementation details including the message exchange protocol/choreography of the interaction.

The BIAN service connection specification is intended to define a business service (i.e. a service connection) in sufficient detail such that a subscriber to that service can switch to an alternative source without suffering excessive business disruption. There will usually be software development/mapping adjustments to make and technical, security and operational considerations to address. The value of BIAN service operation alignment is that these adjustments will hopefully be contained to being local to the service interface itself. The broader (up and down-stream) processing logic dependencies should be stable.

7.1.1 Semantic API Selection Framework

The prior release of this guide defined a framework that can be used to select specific implementation techniques and standards that might apply for the development of an API aligned to a BIAN service connection. A simplified version of this framework is described here.

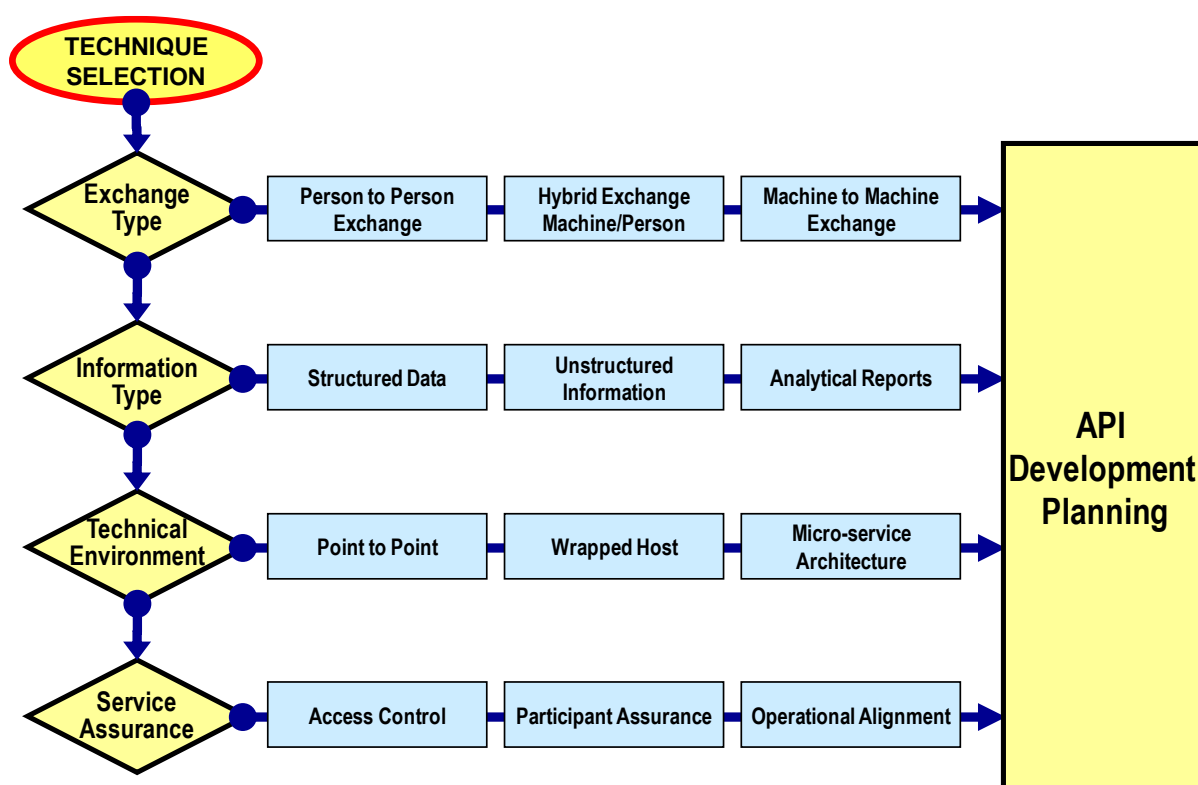


Figure 20: Simplified API Technique Framework

The framework breaks the design consideration into four main categories and a number of techniques (28 in total) are defined under these four main categories:

- *Exchange Type* – recognizes that the implementation requirements will vary greatly depending on the type of entity at either end of the exchange. In particular differentiating between human/cognitive parties and machines. It can be argued that the very definition of an API limits exchanges to automated systems/machines but with many services being presented to mobile devices the definition has been extended to include human readable displays and electronic/smart data collection forms. Furthermore the increased use of machine learning and other AI technology has greatly improved the ability of systems to extract from and interpret information from less structured formats.

The techniques outlined in the attachment covering Exchange type address the different levels of precision needed depending on the type of involved parties. They allow that for cognitive exchanges there is far more latitude for varying formats and content. For example a person can readily browse many different formats and types of content for a bank's product directory/guide where/as agreeing the content of the directory for machine-based extraction would clearly need to be far more restrictive and rigorous.

- *Information Type* – considerations here address the different types of information that can make up the service 'payload'. Different techniques apply for supporting the mapping and exchange of various types of message content. In particular a distinction is made between structured information (both individual information items and structured records) and unstructured information that can take on many forms and media. The recognized information types are: *items* that refer to single value elements such as a date of birth or account balance; *records* that refer structured collections of information items such as a payment transaction or account statement; and *reports* that refers to a wide range of unstructured information as might be found in a free-form report or scanned images.

An additional information type *Analytical views* is defined to allow for a Service Domain to maintain historical and analytical views of individual records or the portfolio of all active records. For example, the 'Current Account Fulfillment' Service Domain may keep track of average balances over a period for a current account and might also provide make-up analysis of the overall portfolio of active current account facilities.

- *Deployment Environment* – recognizes that the implementation of the service operation through an API will require very different design and implementation approaches for different technical production platforms and communication networks. Different approaches are required to broadly align with the three types of API solution approaches already described in

this guide – direct host access, wrapped/ESB host access and container/micro-service based architectures as may be found in highly distributed environments such as the cloud.

- *Service Assurance* – the final category of considerations relates to security and assurance. There are a range of topics to consider including the definition of Service Level Agreements, security profiles and authorization and assurances that may depend on the business relationships and communications environments

A summary of the techniques described for the four main categories is shown – the techniques described in more detail in a prior version of this guide. It provides a description, explains how the technique relates to the BIAN standard and lists any applicable standards and tools/techniques that have been identified at this time.

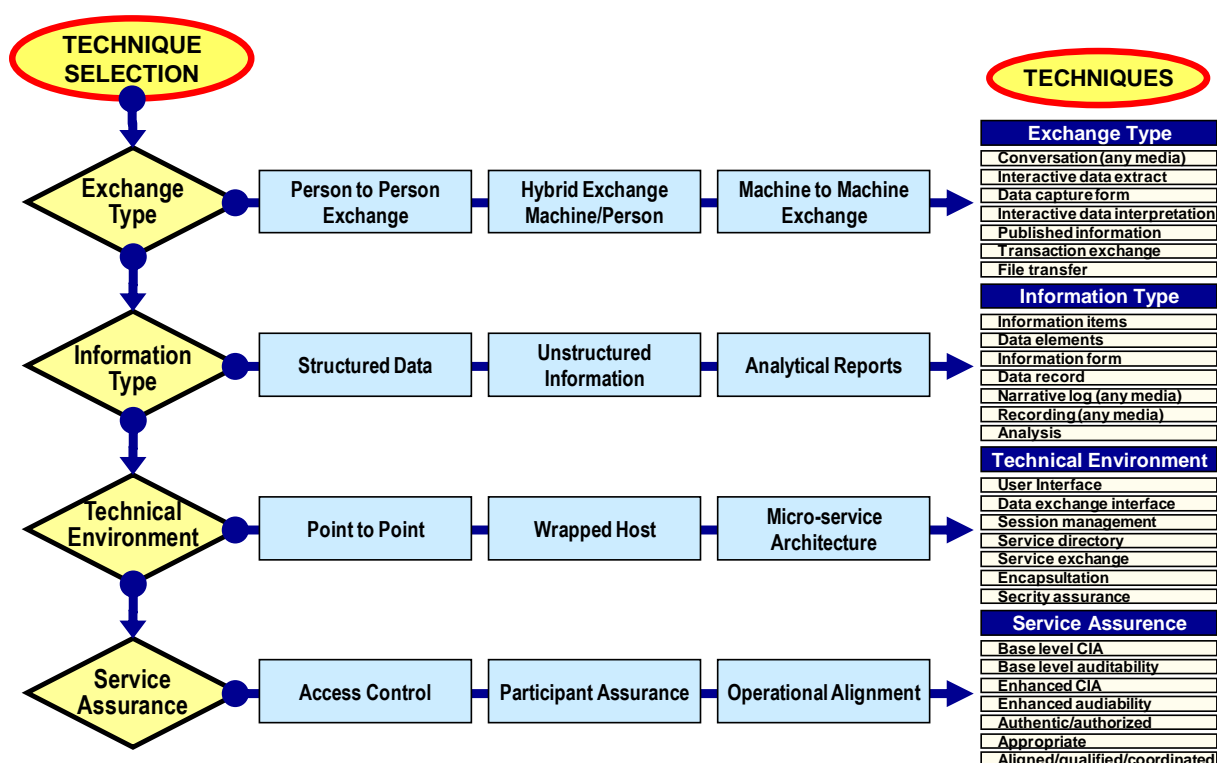


Figure 21: List of 28 Techniques Under the 4 Categories

7.1.2. Service Domain Cluster

A 'cluster' is the term used in BIAN to refer to a related grouping of Service Domains. The grouping may represent either a segment as defined by TOGAF such as the scope of a business unit, profit center, division or enterprise. Or grouping for systems purposes most commonly to reflect the scope of a business system or IT application.

In addition to defining Service Domains and their offered service operations, the BIAN standard captures a growing proportion of the typical service connections between Service Domains needed to address identified business events and requirements are might be represented in a BIAN Business Scenario (for a specific event) or a BIAN wireframe (covering an area of activity supporting many related business events).

The definition of these connections is an ongoing exercise and is only indicative of sensible connections that support known business behaviors. The collection of service connections is not intended to be prescriptive nor complete as new business models and behaviors can result in new patterns of service connections.

The known connections can be used to provide insights into the likely range of interactions. When a selection of Service Domains is defined to generate a cluster, these known service operation dependencies can be referenced to define the external service boundary of that cluster.

The Service Domains within a cluster tend to play one of three roles in the broader context of the overall enterprise' systems portfolio. The possible roles for a Service Domain in a cluster are:

- **Core** – The Service Domain exists only in whatever this cluster represents. Any and all reference to this Service Domain must be supported by the external service boundary of the cluster. (As must all of its delegated service operation dependencies). An example would be the Current Account fulfillment Service Domain in a Core Banking system
- **Proxy** - Represents a capability that is likely to be repeated in other clusters, and is included in the cluster to provide a local 'view/function'. In such a case. It could be the master version meaning all other instances need to reference this instance for their needs, or it could be a slave, meaning that it needs to synchronize with the master instance elsewhere through suitable 'background' services. A typical example would be Customer Reference Data Management that may be replicated in many physical systems for performance reasons
- **Utility** -, The cluster contains a non-unique instance. But in the case where the local instance operates in a fully standalone manner - it does not need to synchronize or even be aware of other similar SD instances

elsewhere. An example would be the Position Keeping Service Domain that is replicated to provide the 'local' transaction journal operational capability for individual product fulfillment Service Domains

An example of a Cluster corresponding to a core banking system is shown with the different Service Domain roles color-coded:

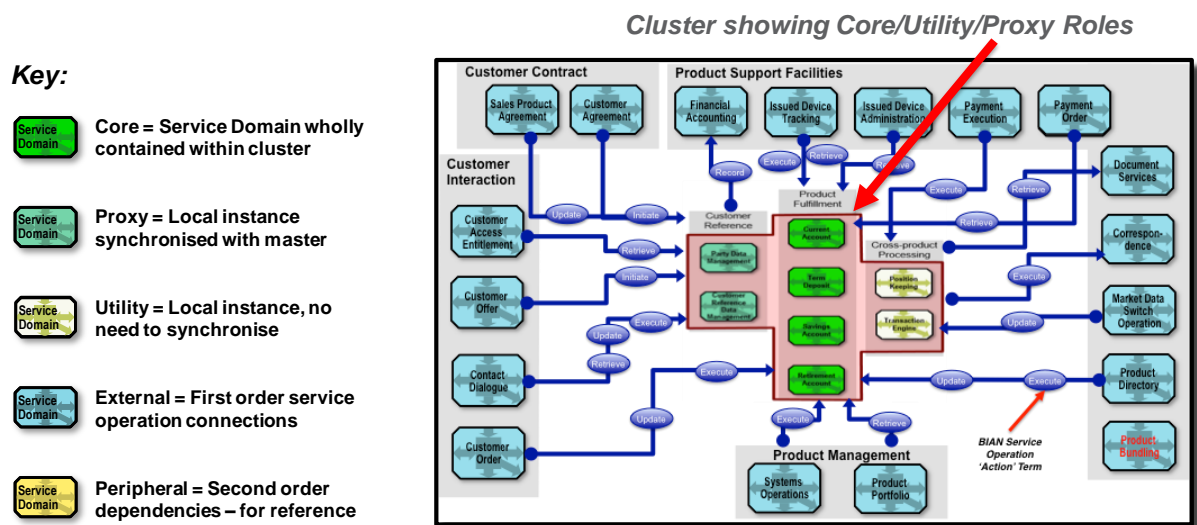


Figure 22: Application Cluster Example

7.2 For Type 1 – Direct to Core

The approach for applying the BIAN standard for direct to core type implementations as noted earlier includes some form of front-end API capability. This implementation/platform specific API facility handles access control (authentication) and physical data import/export facilities for all exchanges.

The BIAN alignment for Type 1 is about how the business service exchange implemented through the API can be mapped as closely as possible to a service connection between two Service Domains. A 'complete' API may group together several business exchanges that map to multiple BIAN service connections. For simplicity we will describe an API that implements the equivalent of a single service connection.

It is assumed for this description that the BIAN example business scenarios, wireframes and extended Service Domain definitions are available for the domain of interest. BIAN is developing these design artifacts across the whole landscape and additional content will be made available as it is developed. (BIAN Members can coordinate to prioritize and develop these designs for areas of specific interest as may be necessary).

The tasks of aligning to the BIAN standard for API development of Type 1 APIs includes the following tasks:

1. *Identify the Business Area/Activity Designs* – select and familiarize with the suitable wireframe, business scenarios and extended Service Domain definitions that relate to the target activity area/domain
2. *Map current physical systems and system boundaries to the Service Domains* – the role of the core host systems (in the context of the API) can be related to the roles of specific Service Domains that are then accessed through appropriate service operations
3. *Isolate the External Access Boundary* – with Direct to Core solutions the access is controlled through the API platform that handles authentication and aspects of the physical data exchange. The link from this managed environment to the associated core system defines the external access path.
4. *Confirm the Mapping* – the BIAN Service Domain, service connection and if necessary wireframe and business scenarios can be referenced to confirm the purpose of the API relates closely to the business purpose of the identified service connection and the mapped host capability
5. *Expand the Semantic Service Operation Specification* – BIAN provides a checklist of the business information that is governed by the offering Service Domain and referenced in its service operations. As noted in some cases these high level semantic attribute lists are mapped to more detailed definitions provided by the BIAN BOM which is an extended version of the ISO20022 Business Model. Where available the designer should reference the RESTful API endpoint specifications that translate the BIAN service operations. These are available through the BIAN API Platform Sandbox
6. *Standard API Implementation* – from this point on the implementation will be determined by physical and practical requirements specific to the implementation and employ established API implementation techniques. The general techniques listed earlier may be referenced to assist with the technical approaches used for the development.

As noted with the latest release a significant sample of the BIAN Service Domains have extended definitions including RESTful endpoint specifications, a proportion of which are already mapped to the extended ISO20022 Business Model. BIAN will be expanding the API related coverage across the remainder of the BIAN Service Landscape as quickly as is practical.

7.3 For Type 2 – Wrapped Host

The approach for a Type 2 wrapped host solution builds on the tasks defined for a type 1 solution. Enterprise Service Bus solutions are often applied across large portions of a bank's application portfolio. The collection of BIAN Service Domains and service operations can be used as a service directory as they define discrete/non-overlapping services. For API solutions the applied ESB solution may be narrow in business scope. As a result some of the usual ESB benefits may be limited.

As with type 1 the core systems are matched to Service Domains and the associated service operations that align to the external access supported by the API and accessed through the ESB are used to define the high-level message content. With the ESB platform additional features can be built into the API solution that improve the performance and/or alignment of the API. The features that should be considered when designing the API were listed earlier in the guide. With some indication of the possible development approach, they are:

- *Host Session Management* – many host interfaces require the set-up of a host access session/log-in for access. The ESB can integrate host access facilities that streamline host session management and possibly enhancing access control/security. Sessions can be persisted between individual service invocations and the implementation of the service can ensure the host access is appropriate for its intended purpose improving access security
- *Data Caching* – The ESB may selectively persist (non-volatile) host data that is accessed frequently to eliminate duplicate host access traffic. Data caching can be self-calibrating/tuning in more sophisticated implementations.
- *Host Wrapping* – the ESB platform may provide an environment to implement wrapping logic that masks shortfalls in the host systems and/or masks host specific features that can be removed from the external API service specification. The function/logic and data wrappers will be specific to the particular hosts system but a key advantage is enable API that align more closely to the standard service definition, eliminating site specific features
- *Resolve Data Fragmentation* – this use of the ESB may be more limited with API solutions as the scope of the ESB may be more limited. In many application portfolios business information will be replicated on many systems leading to problems of fragmentation and inconsistency. An ESB platform can be used to establish the master source of information and coordinate the synchronization with ‘slave’ duplicated views as a background activity
- *Advanced Look-up* – this is a more advanced feature, that can be considered in conjunction with data caching solutions. Access patterns can be defined or ‘learned’ where the subsequent service following a service call can be anticipated and initiated in advance to reduce latency in the host exchange. This feature is useful for more complex, interactive customer dialogues
- *Transaction Persistence* – the ESB can support access transactions that persist and orchestrate processes that can include multiple service exchanges that can access many host systems and involve a series of customer contacts. These facilities quickly start to generate ‘front end’ application capabilities and so should probably be limited/focused in their purpose when implemented within the ESB platform itself

7.4 For type 3 – Distributed Architecture

As noted the Distributed architecture approach is best considered by addressing two distinct aspects. One is the customer access platform. The other includes any core banking activities that may also be supported using a micro-service architecture. Core

banking activities cover a very broad range of activities (essentially all of the Service Landscape beyond the channel access activities that are supported by the customer access platform).

Customer access platform

It is not intended to provide a detailed design of the customer access platform in this guide as this represents a complex solution design that may be tackled many different ways using different technologies and approaches. The key features as defined using the BIAN Service Domains to represent micro-services at a certain level of granularity that can be combined in an integrated solution are outlined as a starting point.

The diagram used to outline the level 3 approach earlier in this guide revealed how the access platform presents a single point of contact to the external user. The platform handles key aspects of the customer interaction including access entitlements, authentication, fraud detection and onward routing decisions. Once these considerations have been addressed the customer contact can be linked through to the bank's core capabilities in a managed/controlled manner. This presents a second single contact point, this time from which to access the range of available core systems. The two contact points are highlighted in the diagram:

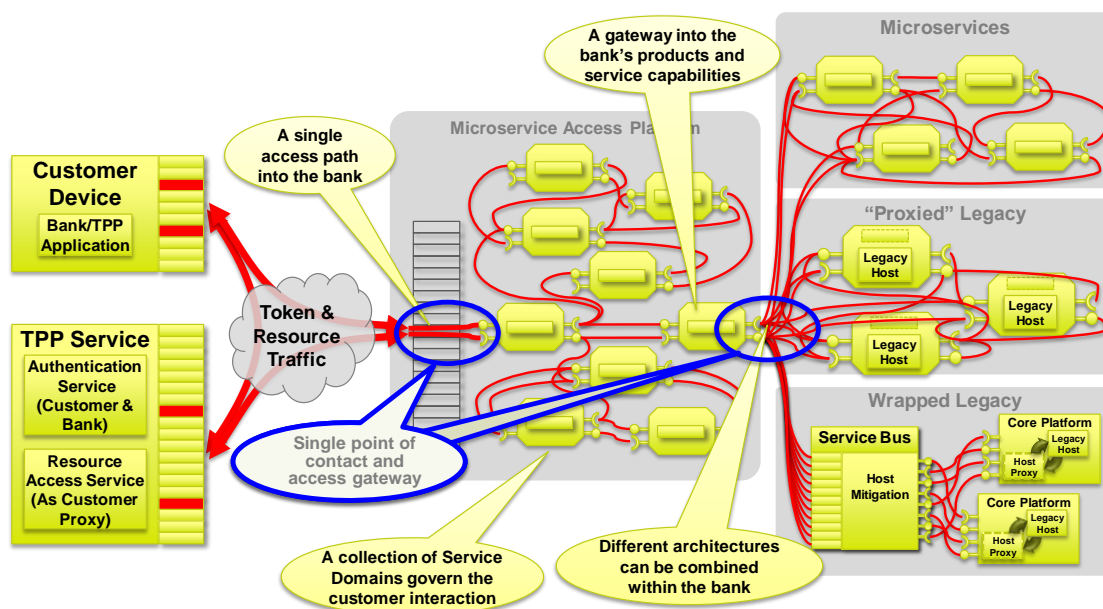


Figure 23: Type 3 With Contact Points Highlighted

The array of capabilities supported by the access platform, particularly when it itself is implemented using a micro-service architecture provides a highly scalable and flexible design. Example business scenarios that define the role and interaction between the various Service Domains can be found in the API Wave 1 designs that are included in the BIAN SL V7.0 release. A simplified wireframe with the key Service Domains is shown here for reference. The main Service Domains have been informally grouped to highlight the different control functions performed by the platform. In the diagram the onward point of access through the 'Contact Dialogue' Service Domain is just one example, in this case a link to the Current Account handling core system (to initiate a payment).

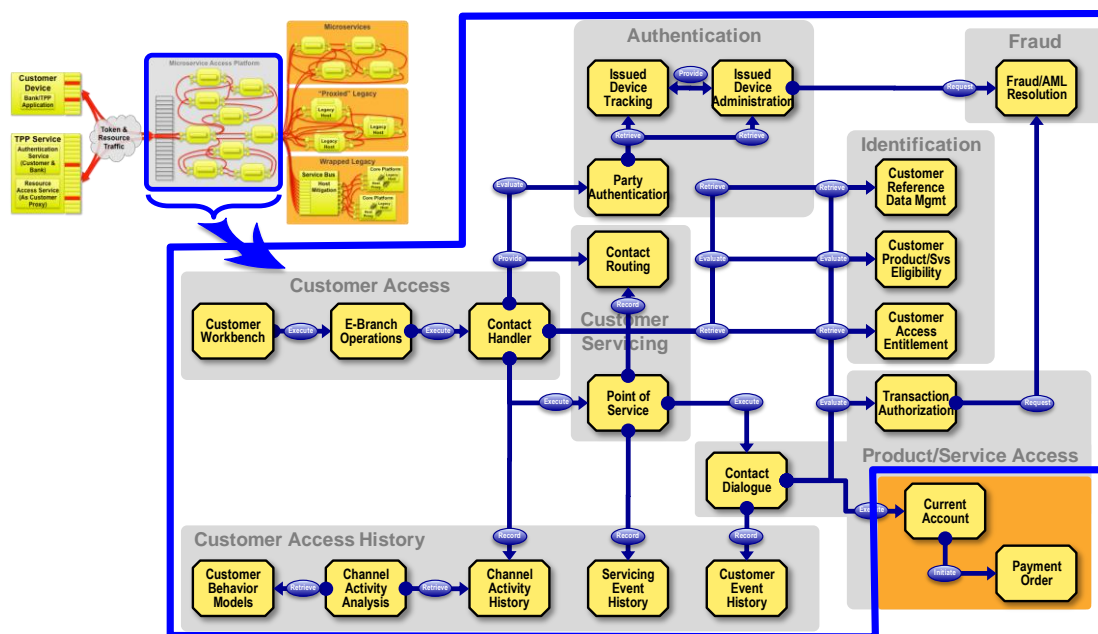


Figure 24: Type 3 - Expanded Customer Access Platform

As noted earlier an advantage of aligning to the BIAN standard at types 2 and 3 is that the same service API can be easily accessed through a customer access platform with limited re-working. The linking *Contact Dialogue* Service Domain that handles the customer interaction can be adapted to reference available type 1 & type 2 services of the core systems as necessary.

The BIAN model is used as a framework to ensure assembled solution components for the access platform are discrete/non-overlapping and implemented to support the maximum flexibility/efficiency. The interactions between the Service Domains/Micro-services will include real-time, nested service exchanges. The Wave 1 content includes a wireframe view that provides an example of how the interactions between the Service Domains might have start/end dependencies that reflects the nesting of their service interactions. This is useful input into a more detailed solution design. An example was shown earlier in this guide (See Table 17)

Core System Micro Service Solutions

The use of a micro-service architecture to implement other banking capabilities should be considered on a case by case basis as type 1-3 based facilities can be combined as necessary to support most types of API requirement. Aligning application boundaries to the BIAN Service Domains results in an application partitioning that conforms to many key micro-service principles as already noted. It eliminates redundancy and through function and information encapsulation results in efficient (and standard) service interactions at the Service Domain level.

A Service Domain can map one to one to a micro-service ‘container’ application, but in practice there may be good design reasons to combine clusters of Service Domains together in an integrated solution. Furthermore a BIAN Service Domain corresponds to a large/coarse grained function partition that is likely to have a complex internal structure that could itself be implemented with an application that uses many finer grained micro-service utilities.

The use of BIAN to model core banking capabilities provides a framework for rationalizing internal application to application (A2A) services by defining non-overlapping business partitions that are well suited to a service based architecture. This can be taken further to define a container partitioning that could be used to implement a micro-service architecture but as noted there may be coarser grained containers to support clusters of Service Domains and a Service Domain container/partition may contain many utility micro-services.

A BIAN Working Group is currently addressing how BIAN Service Domains can relate to a vendor agnostic application architecture that will address how clustering of Service Domains and the support for finer grained utilities is to be handled. For the purposes of API specification it is assumed that the A2A interactions between the bank’s internal applications will align to BIAN Service Domain boundaries such that the service exchanges form the basis for A2A/internal API definition in a similar manner to the design of external/open APIs already discussed

8. BIAN Design Content – SL V 7.0

The BIAN SL V7.0 release includes the Service Landscape API Inventory view as described earlier in this document. The inventory will be expanded as more mapped open API solutions are made available.

The release also contains a collection of extended designs for API development covering a selection of the BIAN Landscape referred to as Wave 1. BIAN will be expanding the extended content to eventually cover the whole Service Landscape in a series of development waves. The initial content will be published initially in 'provisional' form, meaning the designs conform to BIAN design principles and the anticipated functioning of the Service Domain but will not have been used in a production context to ratify their accuracy/completeness.

The BIAN API development approach selects areas of the landscape based on member interest and priorities and then defines extended definitions for that are to provide the high level designs that can be used in API specification. The output of a wave is one or more collections of API solution sets where a solution set addresses a particular business area or function supported by a collection of related APIs.

The outline approach used to develop wave content is shown in the schematic:

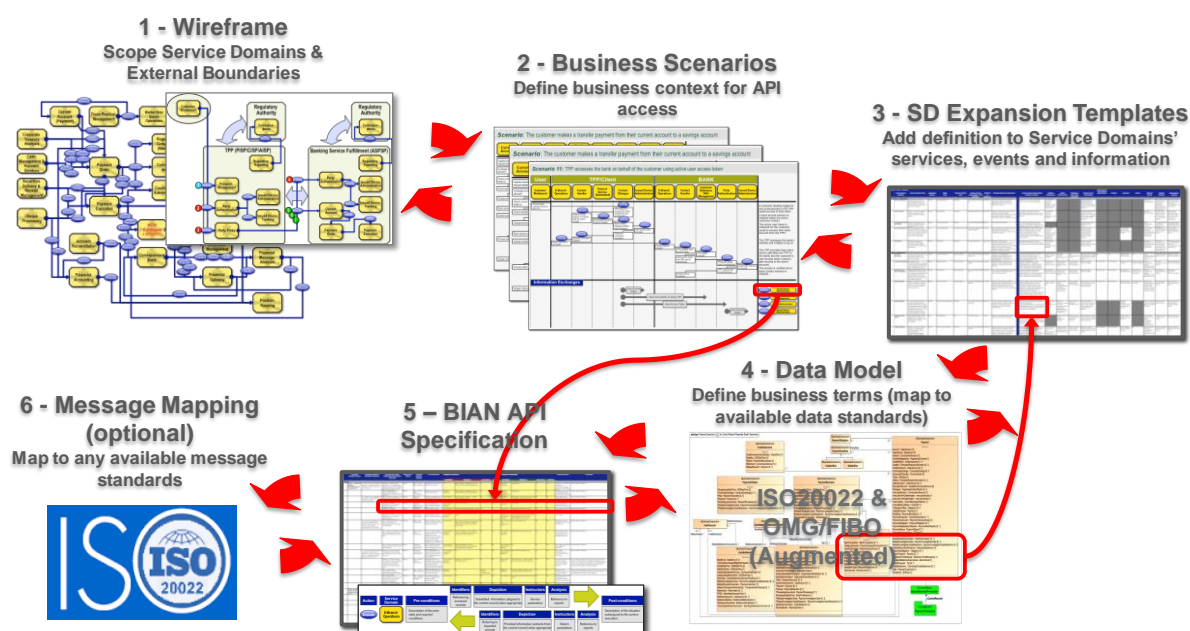


Figure 25: API Content Development Approach

The steps and deliverables are in outline:

1. *Wireframe* – a wireframe shows the involved Service Domains and their main service connections (with associated action term) matched to the Bank, an optional Third Party intermediary and the Customer's own device
2. *Business Scenarios* – a collection of example business events/processes that characterize the function and in particular expose the external service exchanges that are to be supported by the open API
3. *Extended Definitions* – extended definitions of the Service Domains including the specification of their behavior qualifiers and the associated service operations and business information content
4. *ISO20022 Mapping/BOM* – mapping the semantic information of the Service Domain's control record to existing BOM and expanding/mapping to the evolving BIAN BOM as appropriate
5. *Service Operation Definition* – expanding the description of the individual service operation exchanges including purpose, pre and post conditions along with the semantic information exchanged
6. *(optional) Message Mapping* – mapping the service operation exchanges to established industry message formats when available (note: very few industry standard messages are available to support this mapping)

The content for Wave 1, included in the SL V7.0 Release includes wireframes, business scenarios, extended Service Domain definitions, underlying business object model and service operation descriptions for four API solution sets:

1. Mobile Access (& Security) – a basic customer access platform
2. Customer On-boarding & Offer Handling
3. Payments (mainstream PSD2 requirements)
4. Consumer Loans – a simple unsecured loan

The complete specifications can be found with the SL V7.0 release. Example outputs are included as an attachment to this guide for the Customer On-boarding & Offer Handling API solution set including:

- The wireframe
- Business Scenarios
 - Prospect On-boarding (KYC)
 - Suitability/Eligibility Checks
 - Configuration and Pricing Negotiation
 - Credit/Risk/Underwriting Decisions
 - Documentation & Compliance Checks
 - Product Booking, Recording and Set-up Initiation
- Extended Service Domain Excel template
- Example service operation Excel template

Upcoming Waves will expand the product coverage re-using the more general mobile access and on-boarding capabilities provided in the first wave.

Attachments - Example Wave 1 Deliverables

The wireframe

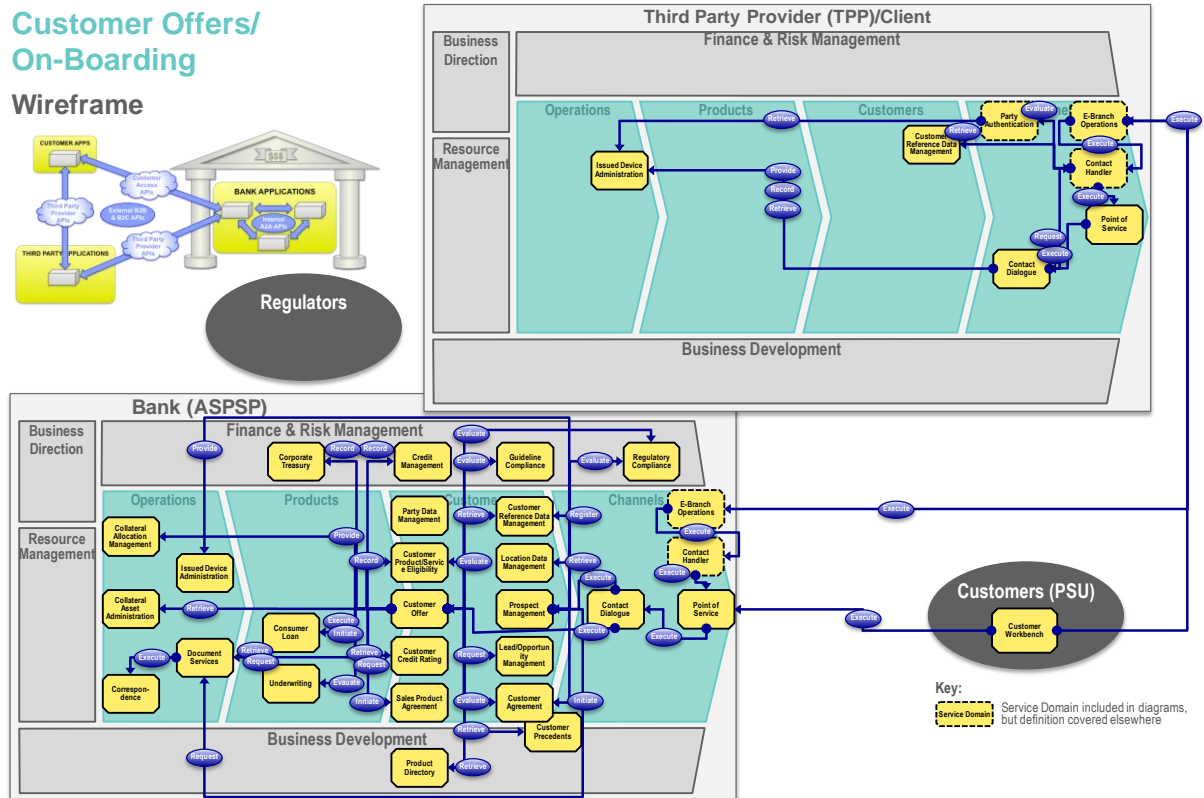


Figure 26: Wireframe example - Customer offers / onboarding

Business Scenarios

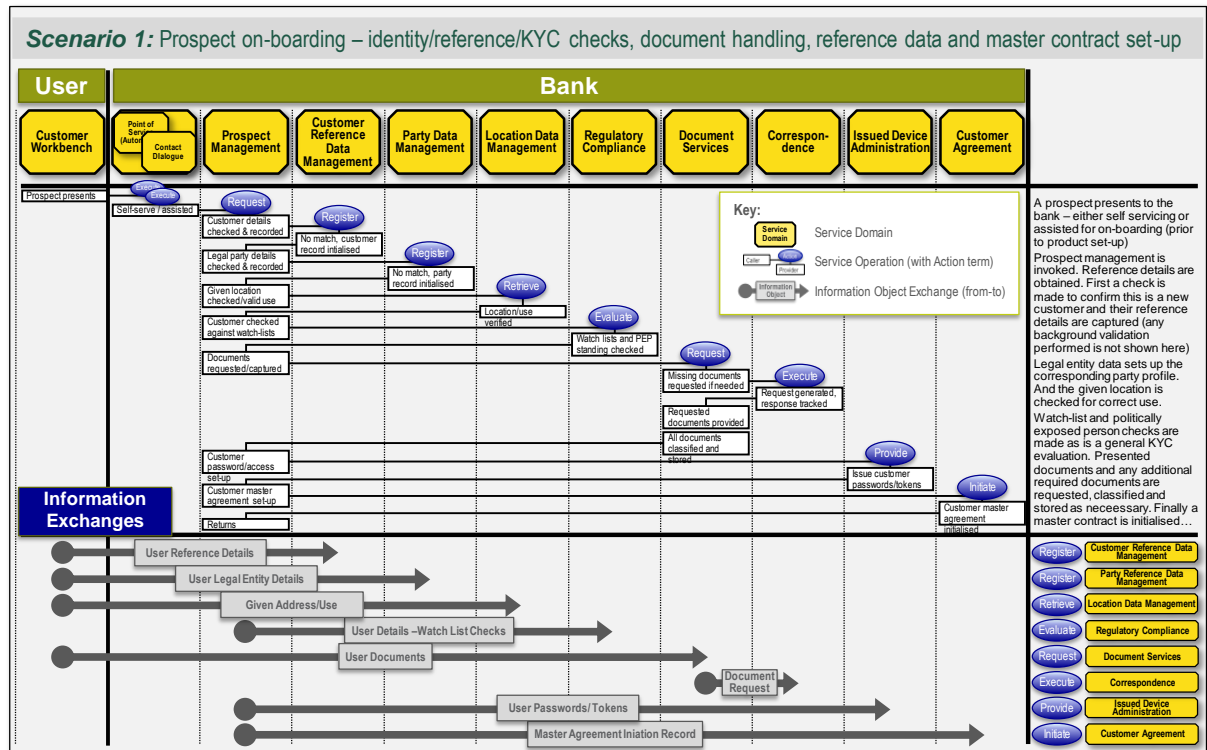


Figure 27: Prospect On-boarding (KYC)

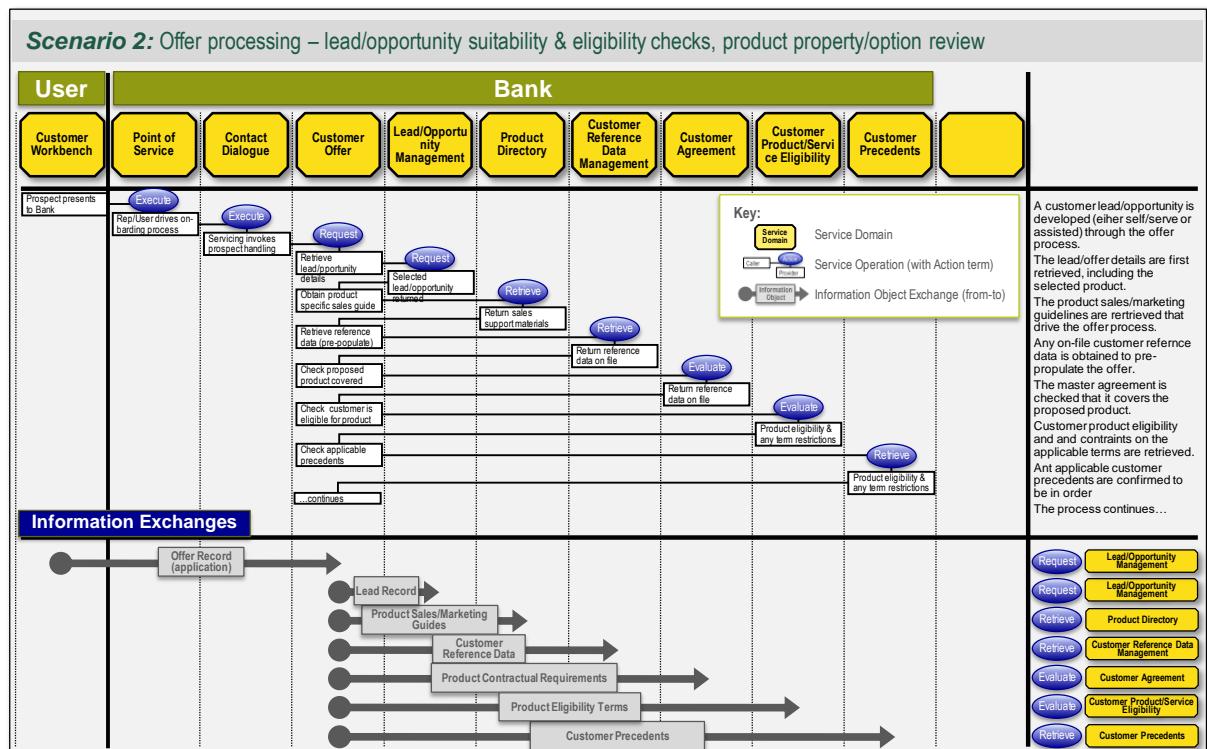


Figure 28: Suitability/Eligibility Checks

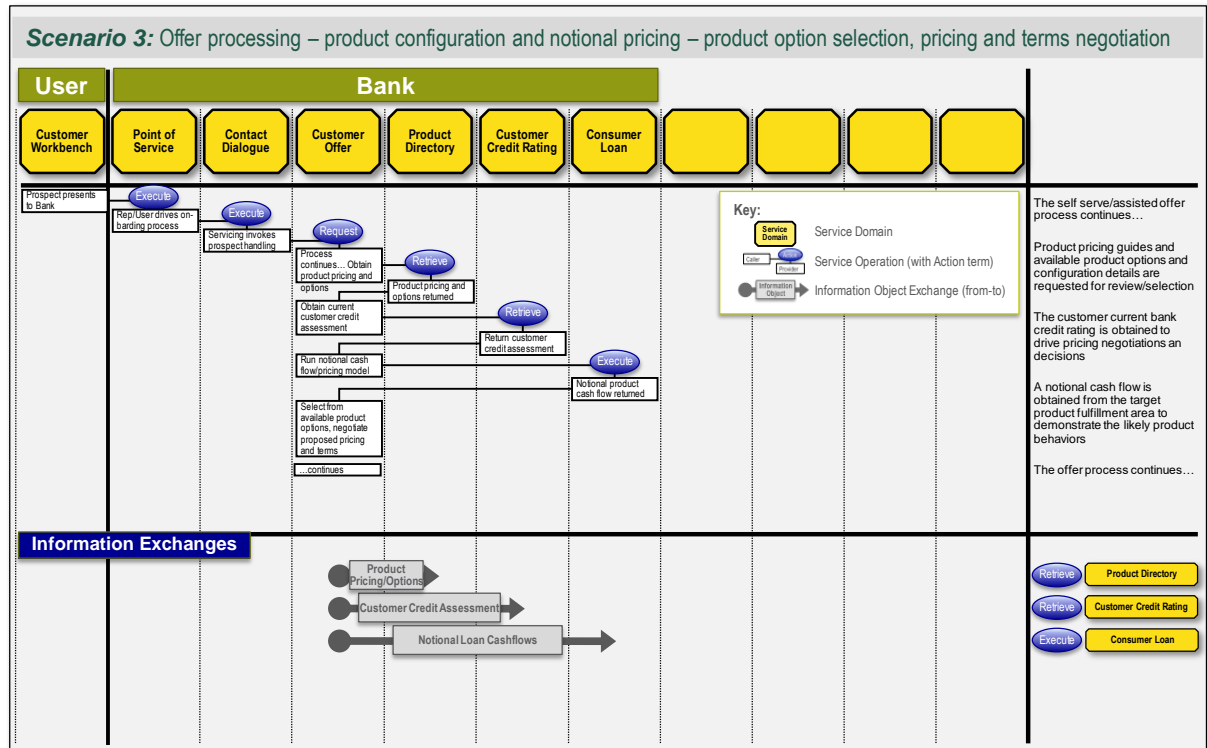


Figure 29: Configuration and Pricing Negotiation

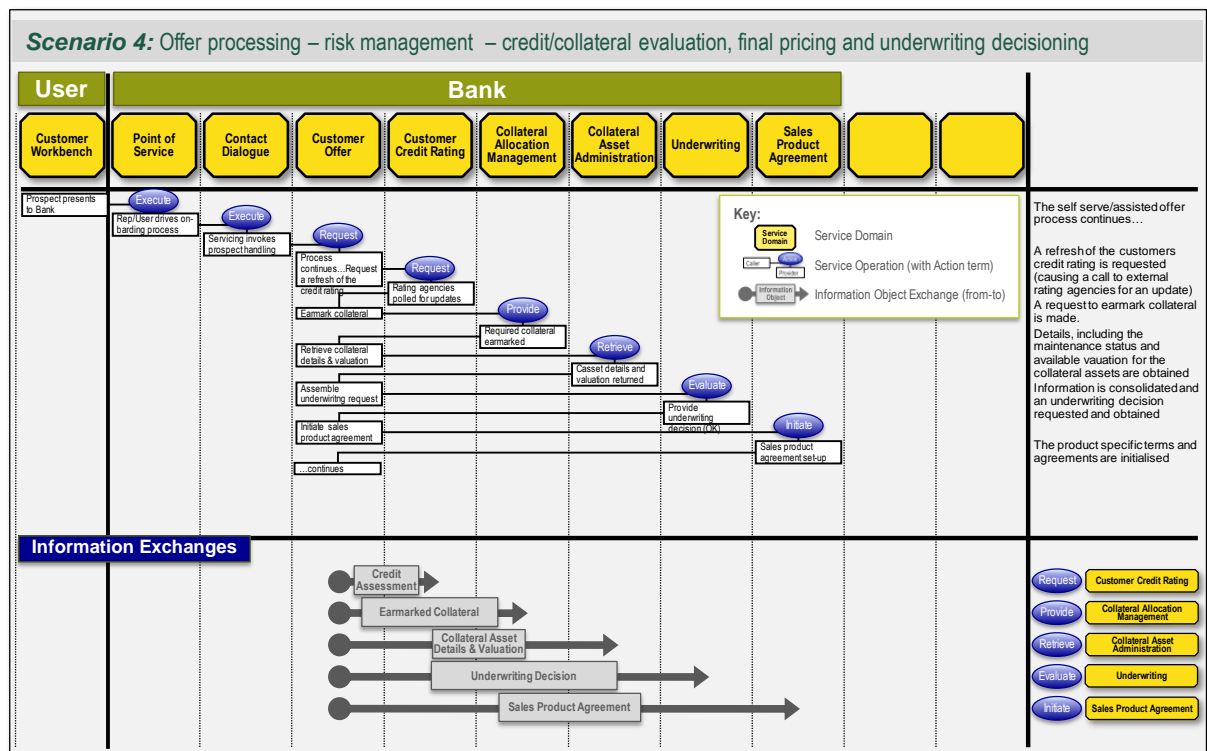


Figure 30: Credit/Risk/Underwriting Decisions

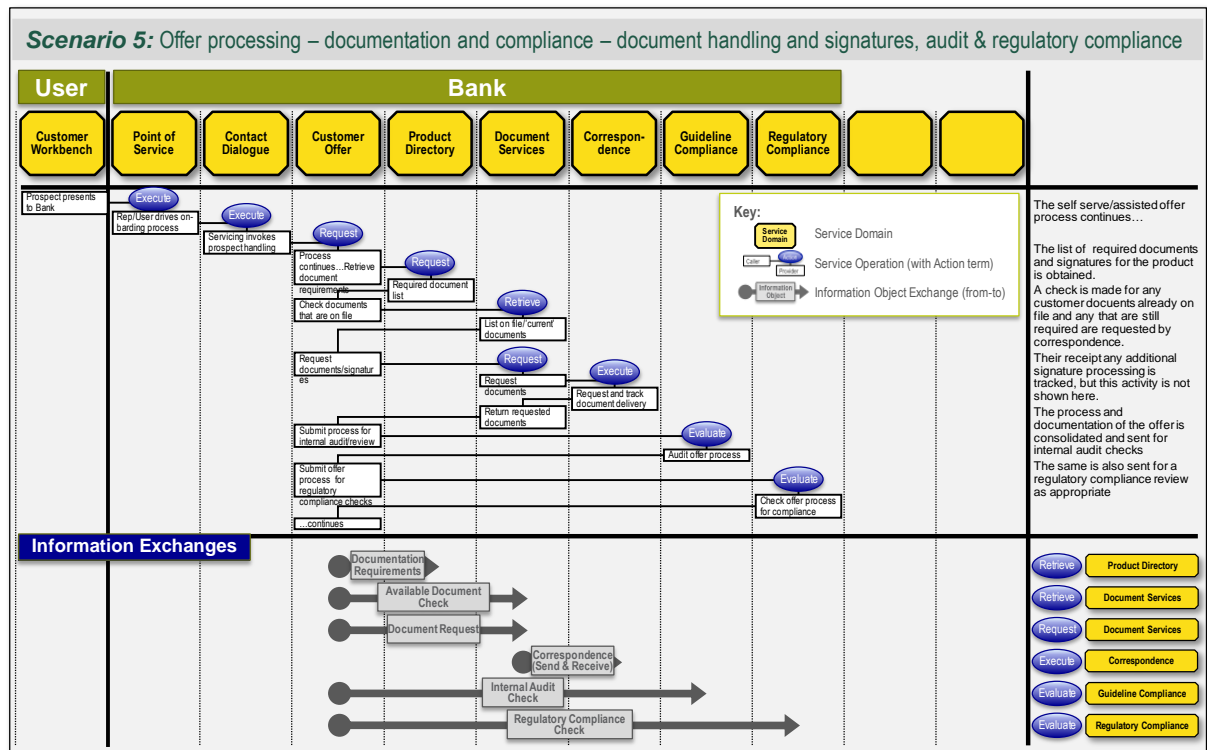


Figure 31: Documentation & Compliance Checks

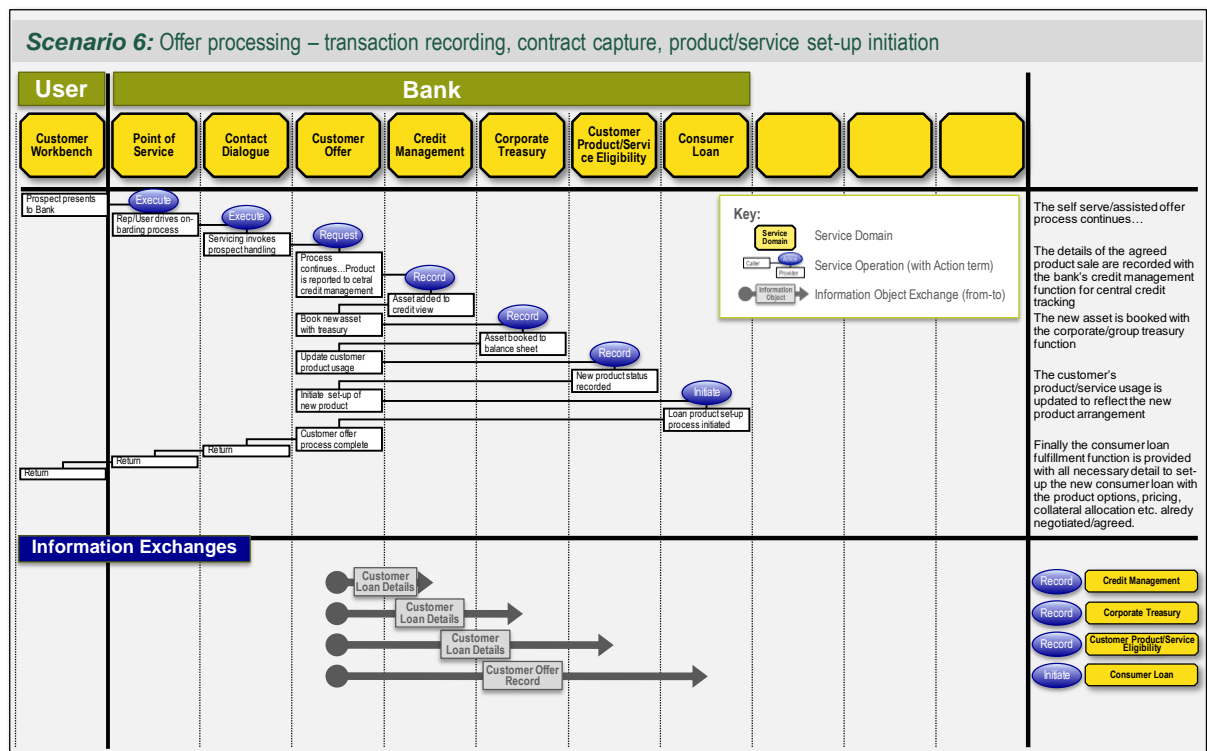


Figure 32: Product Booking, Recording and Set-up Initiation