Banking Industry
Architecture Network

BIAN
How-to Guide

Applying the BIAN Standard

Organization

Authors		
Role	**Name**	**Company**
BIAN Architect	Guy Rackham	BIAN

Status			
Status	**Date**	**Actor**	**Comment / Reference**
DRAFT	October 2018	Guy Rackham	Reworked
Approved		Architectural Committee	

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

Copyright

Table of Contents

Table of Figures

1 BIAN How-to Guide – Applying the BIAN Standard

1.1 Document Introduction

The BIAN standard defines generic business functional capacity partitions (Service Domains) and their semantic service operations. In order to map these standard designs to a specific organization they need to be selected, adapted and assembled to match the operational scope and structure of the organization and its underlying business applications. BIAN's high level conceptual definitions must then be mapped to more detailed implementation level technical designs. This third document of the BIAN How-to Guide presents the current guidelines for applying the BIAN designs in different business and technical environments and situations.

This document is continually revised to reflect deployment insights gained between the major Service Landscape release cycles. With this release an update to a related guide covering the use of BIAN to support API development – The BIAN Semantic API How to Guide has also been produced. Extracts of that updated guide are included in this guide for ease of reference.

1.2 BIAN How-to Guide – Applying the BIAN Standard

This final document of the BIAN How-to Guide series explains how the BIAN standard can be used in deployment. As BIAN rapidly adds content to the model more experience is gained and new approaches are developed that are reflected back into these guidelines. The guidelines outlined in this document present the current view on different possible deployment approaches. These and new approaches will be refined and expanded as BIAN and BIAN members use the standard.

Since the last version of the How-to Guide there has been a strong focus on the use of the BIAN model to support the definition of APIs. This has included the set-up of the BIAN API Exchange. The API Exchange contains RESTful API specifications derived from the offered service operation definitions of 67 selected Service Domains. BIAN will be expanding the coverage of the Exchange as fast as is practical. As noted a specific How to Guide addresses the specific use of BIAN for developing APIs. This guide summarizes the key content of the API guide, but also covers other possible applications of the BIAN standard. Given the recent focus on API development there are only limited additions to this version of the guide.

These guidelines have benefited from several significant implementation projects and initiatives over recent years that have leveraged and extended the BIAN model. For some of these projects related case studies and white papers can be found on BIAN.org. Specific additions made with the latest release are:

1. The specification of the Service Domain and its service operations has been further extended and more formally structured in order to support the BIAN semantic API Initiative.
2. BIAN has embarked on defining the BIAN Business Object Model (BOM). This effort is also closely integrated with the BIAN API initiative. The BIAN BOM builds on the ISO20022 Business Model, adding extensions as needed to align to the BIAN Service Domain and service operation structures and to add content that is not fully represented in the ISO model
3. BIAN Business Capability Model – the latest release includes a first version of the BIAN Business Capability model. However this model will not be mapped to the BIAN service domains until the level three capability definitions are complete

In addition there are significant on-going activities within BIAN that are likely to be reflected in the next cycle or the guides. These include:

1. Further definition of a business capability view of the BIAN Service Landscape – this view will help business practitioners access the standard when the roles of the Service Domains are not intuitive and also provides a way for associating business value and performance in the context of specific business capabilities
2. The definition of a vendor agnostic application architecture view of the Service Landscape – this view will provide a structured way to map the business level BIAN designs to the various aspects of more detailed application architectures in a way that is agnostic to any one particular implementation, but also that supports traceability between physical solutions

The intended audience for this document is the business and technical architects of the BIAN membership and any individual or organization seeking to apply the BIAN designs in practice. As with other documents of the How-to Guide series, some of the topics covered here from a deployment point of view are revisited in the other documents of the How-to Guide from their respective viewpoints.

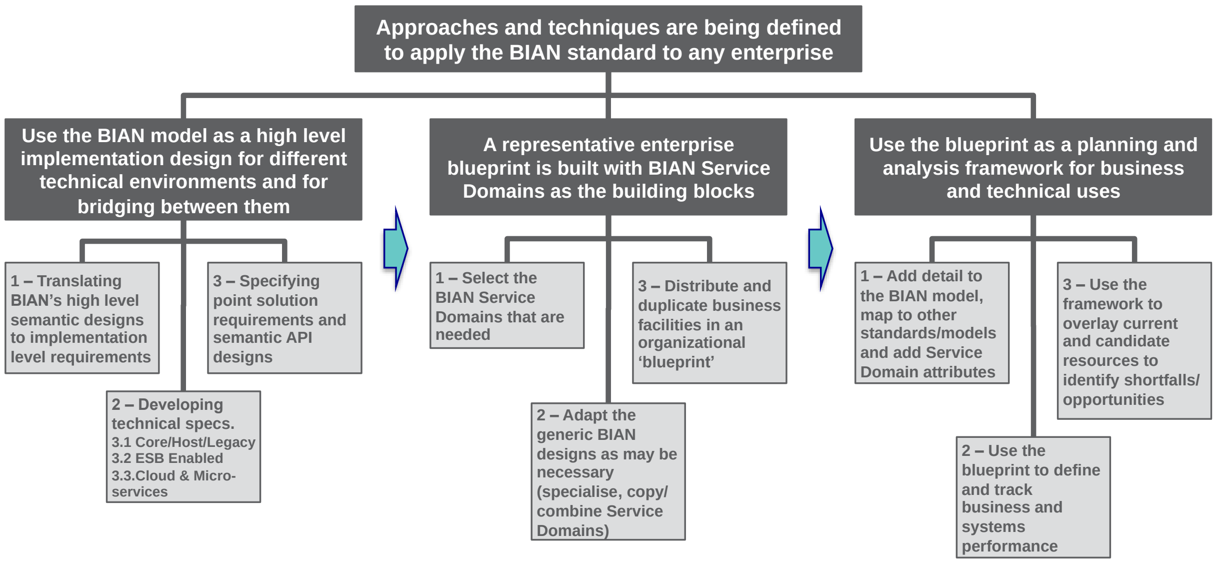

Approaches and techniques are being defined to apply the BIAN standard to any enterprise
Use the BIAN model as a high level implementation design for different technical environments and for bridging between them
1 – Translating BIAN's high level semantic designs to implementation level requirements
3 – Specifying point solution requirements and semantic API designs
2 – Developing technical specs.
3.1 Core/Host/Legacy
3.2 ESB Enabled
3.3.Cloud & Micro-services
A representative enterprise blueprint is built with BIAN Service Domains as the building blocks
1 – Select the BIAN Service Domains that are needed
3 – Distribute and duplicate business facilities in an organizational 'blueprint'
2 – Adapt the generic BIAN designs as may be necessary (specialise, copy/ combine Service Domains)
Use the blueprint as a planning and analysis framework for business and technical uses
1 – Add detail to the BIAN model, map to other standards/models and add Service Domain attributes
3 – Use the framework to overlay current and candidate resources to identify shortfalls/ opportunities
2 – Use the blueprint to define and track business and systems performance

Figure 1: Applying the BIAN standard content

As can be seen in the above overview Figure, the deployment approaches are explained in three main sections.

Using the BIAN model as a high-level implementation design – the BIAN business architecture model needs to be related to more detailed systems architecture views for implementation. This complex topic is broken down as follows:

1. SOA – the benefits and stages of adoption
2. Relating the BIAN business model view to business applications
3. Service Domain clusters
4. Adding detail to the BIAN business architecture specification
5. Mapping the BIAN Service Domain in different technical environments
   - Type 1 - Conventional (legacy/core) system rationalization
   - Type 2 - Host renewal/ESB integration
   - Type 3 - Loose coupled distributed/cloud and micro-service architectures
6. Point Solutions – steps and templates used to apply the BIAN designs
7. Semantic API designs

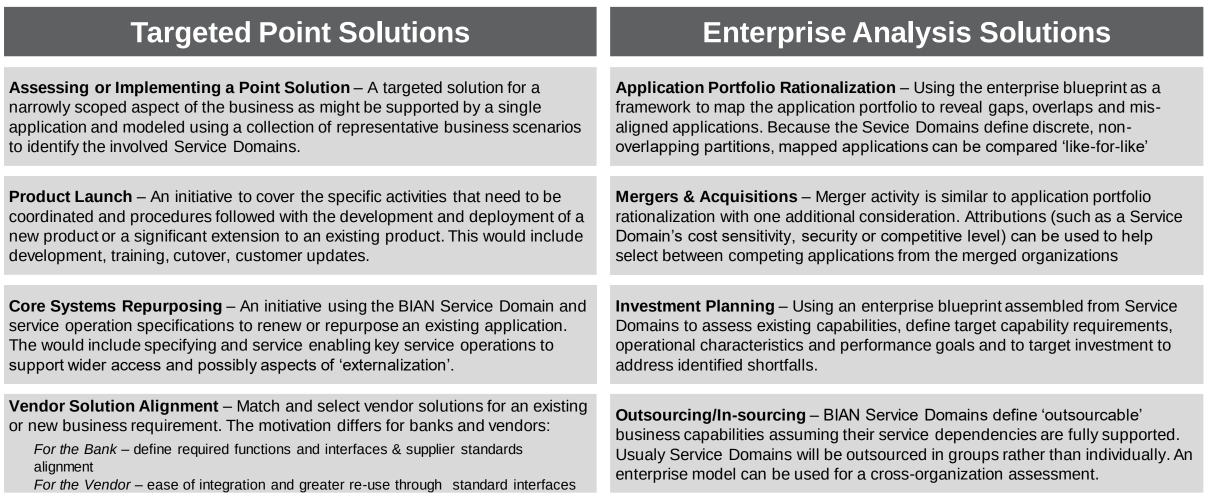
Building an Enterprise Blueprint – the BIAN Service Landscape contains one and only one of each identified Service Domain in a 'reference framework'. The Service Domains can be thought of as 'building blocks'. In order to assemble these building blocks into a representative model of a specific enterprise – the enterprise blueprint - three steps are defined:

1. Select/filter Service Domains to match the range of activities at the enterprise.
2. Specialize/adapt service domains to reflect specific needs/behaviors of the enterprise.
3. Duplicate and arrange Service Domains to match the organizational structure of the enterprise.

An enterprise blueprint contains the selected Service Domains some time duplicated and then set out in a structure reflecting the structural make-up of the enterprise. This includes the way the business chooses to segment the market. The way the BIAN Service Domains can align to different types of bank and their associated market segmentation is discussed

Using the Enterprise Blueprint for Planning & Analysis – because the BIAN Service Domains define business roles that are highly enduring ('what' they do does not change, 'how', 'when' and perhaps 'why' will change as business practices and solutions evolve) an enterprise blueprint assembled using Service Domains is highly stable over time. As a result such a blueprint provides an excellent framework suited to a wide range of planning and analysis activities

In past internal BIAN discussions and earlier versions of the How-to Guide a number of general types of projects or initiatives were identified that could leverage the BIAN standard. The types of projects aligned to the two general categories of deployment, as outlined in the next Figure:



Targeted Point Solutions	Enterprise Analysis Solutions
Assessing or Implementing a Point Solution – A targeted solution for a narrowly scoped aspect of the business as might be supported by a single application and modeled using a collection of representative business scenarios to identify the involved Service Domains.	**Application Portfolio Rationalization** – Using the enterprise blueprint as a framework to map the application portfolio to reveal gaps, overlaps and mis-aligned applications. Because the Sevice Domains define discrete, non-overlapping partitions, mapped applications can be compared 'like-for-like'
Product Launch – An initiative to cover the specific activities that need to be coordinated and procedures followed with the development and deployment of a new product or a significant extension to an existing product. This would include development, training, cutover, customer updates.	**Mergers & Acquisitions** – Merger activity is similar to application portfolio rationalization with one additional consideration. Attributions (such as a Service Domain's cost sensitivity, security or competitive level) can be used to help select between competing applications from the merged organizations
Core Systems Repurposing – An initiative using the BIAN Service Domain and service operation specifications to renew or repurpose an existing application. The would include specifying and service enabling key service operations to support wider access and possibly aspects of 'externalization'.	**Investment Planning** – Using an enterprise blueprint assembled from Service Domains to assess existing capabilities, define target capability requirements, operational characteristics and performance goals and to target investment to address identified shortfalls.
Vendor Solution Alignment – Match and select vendor solutions for an existing or new business requirement. The motivation differs for banks and vendors: *For the Bank* – define required functions and interfaces & supplier standards alignment *For the Vendor* – ease of integration and greater re-use through standard interfaces	**Outsourcing/In-sourcing** – BIAN Service Domains define 'outsourcable' business capabilities assuming their service dependencies are fully supported. Usualy Service Domains will be outsourced in groups rather than individually. An enterprise model can be used for a cross-organization assessment.

Figure 2: Projects split between point and enterprise solutions

The 'point solutions' are addressed by the range of topics covered in the first main section of this guide: "Using the BIAN model as a high level implementation design". The enterprise solutions are then covered in the second and third sections – "Building an Enterprise Blueprint" and "Using the Enterprise Blueprint for Planning & Analysis".

As BIAN members undertake implementation projects leveraging the standard BIAN will continue to provide case studies when possible for review at www.BIAN.org and the experiences gained will be used to continually expand and refine these deployment guidelines and other more specific guidelines as necessary.

2 Using BIAN Specifications as a high-level Implementation Design

BIAN designs can be extended and used to define systems requirements for many types of solution implementation projects. The BIAN standard is a business architecture level model that defines a type of service-oriented architecture (SOA). A SOA captures the business activity as a collection of collaborating operational service centers. It might be expected that the only type of systems architecture that could be linked or derived using the BIAN model would correspondingly be service oriented.

There are several significant operational advantages in service based systems design. But the BIAN business architecture provides valuable insights and design structures for most of the prevailing technical environments found in banks (as described in more detail below).

This section addresses the considerations and approaches for interpreting the BIAN standard in solution design and implementation. It is structured into a number of sub-sections as follows:

1. **SOA – benefits & ‘externalization’** – there are benefits for adopting service based designs at the technical systems level and at the higher business architecture level defined by BIAN – these are outlined. The benefits can be associated with the degree or level to which the service oriented concepts are adopted in the application architecture. In this guide we informally consider three stages/levels of adoption. These levels are used to explain an important BIAN concept of ‘externalization’ which is key to ensure Service Domains enforce good data and function encapsulation.
2. **Business to Technical Architecture – Mapping Service Domains** – the BIAN Service Domain is a conceptual design of a business capability partition that is defined in terms of its business function and the service operations it offers and consumes. This business capability partition can be mapped to the supporting business applications and physical systems in various ways
3. **Service Domain Clusters** – a ‘cluster’ represents a collection of Service Domains as might map to a business application. Different roles for the contained Service Domains are defined to help manage the service dependencies that define the external boundary of the application
4. **Adding detail to the BIAN business architecture specification** – the BIAN standard and supporting artifacts provide a high level specification of the core functionality, business information use and service operation boundary of Service Domains. With the prior release (V6.0) BIAN first introduced an additional level of specification to the Service Domains. The business architecture specifications provide an organizing framework for adding the additional layers of detail needed to specify systems requirements and implementation designs. These

layers of detail can be considered in terms of application logic, information/data and communications.

5. **Interpreting SD's in different technical environments** – as noted earlier, in the How-to Guide series we define three informal stages/levels of SOA adoption. These indicative levels have been used to consider how the BIAN standard applies in the three main, fundamentally different prevailing technical architectures found in most banks today
6. **Point Solutions** – sets out the general steps that can be followed when leveraging the BIAN business architecture in the context of a 'point solution'. This includes describing some working templates and model views that have been used in recent BIAN implementation projects
7. **Semantic API Initiative** – this initiative use extended BIAN specifications to define high-level API designs. These designs can be applied at levels of sophistication corresponding to the three prevailing technical architectures described earlier

2.1 Service Oriented Architectures & the Benefits of 'Externalization'

The benefits of adopting service oriented architecture (SOA) approaches in systems design and implementation are well understood documented. In the How To Guide – Creating Content the generally accepted benefits and those more specifically addressing the BIAN approach are referenced. They are summarized here for quick reference.

The general IT systems related benefits for adopting SOA as described in detail by the Open Group can be paraphrased as follows:

- *Service* – the adoption of services in the systems architecture can improve information flow, help expose embedded functionality and offer greater organizational flexibility.

- *Service re-use* – service based software leads to lower software development and management cost.

- *Messaging* – has a wide range of positive impacts including configuration flexibility, better monitoring and intelligence, greater control and security.

- *Complexity and Composition* – services can simplify software supporting more complex, adaptive and more easily integrated solutions.

The SOA benefits described by the Open Group relate to the impact on the development, performance and fit-to-purpose quality of software solutions. BIAN applies the SOA concepts at the level of business architecture – defining the operational capability partitions and interactions that characterize operating practices within the bank rather than the specific mechanics of their supporting systems.

Some of the key business architectural design properties that BIAN implements include:

- *BIAN Service Partitions are Discrete* – the business purpose of a service partition is unique, non-overlapping and discrete.

- *BIAN Service Partitions are collectively comprehensive* – BIAN seeks to define a complete set of service partitions. All possible banking activity can be modeled using the identified Service Domains

- *BIAN Service Partitions are 'elemental'* – the Service Domain supports a single business purpose. They are not made up of smaller service domains, instead the collection of identified Service Domains forms a 'peer set'.

As a result of these specific operational design properties the BIAN SOA provides additional opportunities when used better to align the underlying business applications:

- *Operational re-use:* the unique operational capabilities of individual Service Domains can be widely accessed across the enterprise increasing operational capability re-use, concentrating scarce and/or specialized resources and improving resource utilization/leverage.

- *Increased operational flexibility*: as more business functions are made available through shared services, changing business needs and operating business models can more readily be supported through service realignment/re-use. In time these can in cases and where appropriate be offered by external parties

- *Reduced business information inconsistencies and fragmentation:* the SOA partitions act as the single source for the business information that they 'govern'. This property is used to reduce inconsistency and fragmentation as Service Domains maintain an autonomous/encapsulated view of their own business information.

- *Performance optimization*: each service partition fulfils a narrowly defined business purpose so its internal capabilities can be optimized for that specific behavior

- *Support for distributed systems solutions* – because the Service Domains define discrete business functional capacity partitions that fulfill the full life-cycle of their role they define highly encapsulated entities. These partitions are well suited for distributed environments such as the cloud and micro-service architectures where access to a shared/centralized database is not always a practical option

The building block of the BIAN SOA is the Service Domain – it is a conceptual specification of a functional partition. A critical aspect of the Service Domain's definition is to ensure effective encapsulation. In order to define properly encapsulated designs it is important to clearly distinguish between functions that a Service Domain performs directly (using its own internal processing logic) and functions for which it still retains the ultimate delivery responsibility but that it relies on other Service Domains to execute through making delegated service calls. The design approach to determining what functionality should be delegated is referred to as 'externalization' within BIAN.

Defining BIAN's 'Externalization' approach

Externalization is an approach used to determine what a Service Domain should do itself and when it should call on or 'delegate to' the services of another Service Domain. Externalization ensures that each Service Domain performs a single discrete function and so enforces the encapsulation principle.

The way a Service Domains is scoped out is described in detail in the How-to Guide – Design Principles & Techniques. In summary a Service Domain's business purpose or role combines the enforcement of a type of commercial behavior ('functional pattern') that it applies to instances of a type of asset. This role is characterized by the Service Domain's 'control record' – a mechanism that it uses to keep track every time it performs its role from start to finish or its complete life-cycle.

For example, there is an 'Employee Assignment' Service Domain. Its associated commercial behavior is assigning work and the asset is the employee (actually the employee's work capacity to be precise). The Service Domain covers the processing logic and governs the business information needed to handle all work assignments through their full life-cycle. A single control record instance is used to capture, track and report on an individual employee's work assignments.

In order to fulfill its business role a Service Domain may need to call on a wide range of other specialized Service Domains for many different reasons. For example, the Employee Assignment Service Domain may need to check the employee's qualifications for a proposed assignment. Employee certification is a different specialized function. The Employee Assignment Service Domain delegates the employee's certification assessment to another Service Domain – i.e. the certification function is 'externalized' for the Employee Assignment Service Domain.

In summary the functionality contained within and the business information governed by the Service Domain needs to be limited to the logic and information needed to address the life-cycle of all instances of its own control record directly. Any other functionality should be external, i.e. accessed through delegated services that call on some other suitable Service Domain.

The concept of externalization can be clarified by contrasting it with more conventional sub-routine calls that behave in a similar way but are not used specifically to enforce proper encapsulation:

- **Responsibility allocation** – responsibility is specifically allocated with an externalized service call as follows: the responsibility for confirming that the call is appropriate in the first place, subsequently making the call, accepting and acting on the result all remains with the delegating Service Domain. The responsibility of the called service provider is only to deliver to the actual or implied service agreement associated with the service operation.

  Assigning responsibility in a delegated exchange is an important aspect of service design and is necessary to protect the principle of encapsulation. A service provider controls the delivery of the service offered. They must make clear the nature/performance properties of the service they offer in order for the service consumer to make an accurate decision on the suitability of the service for their particular need. The service consumer retains responsibility for their decision to use a particular service and for their acceptance and application the returned result of the service call.

  For example a person that uses a taxi service to get to the airport can reasonably expect that the taxi is well maintained and fueled-up. But what if traffic is particularly bad, or the taxi gets involved in an accident or suffers a flat tire and the individual misses the flight? Applying the definition of externalization the fault for missing the flight is with the decision to use the taxi service (with insufficient contingency) and not with the taxi service itself.

  The allocation of responsibility with utility calls is not necessarily so explicit. Users and allowed/intended usage is not as well assigned if at all as they are in the service based model.

- ***Business Information/Data Access*** – for a delegated service there is an implicit assumption that all information/data that needs to be agreed between the parties to fulfill the service exchange is contained in the exchanged messages underlying the exchange. Conversely with process/utility calls there can be assumptions made that there is some shared/global database with common data definitions available to both involved parties in order to support the interaction.

  Note that the concept that each Service Domain is responsible for its own autonomous internal ‘datastore’ and only needs to agree definitions of the information exposed through service operation exchanges is another key facet of encapsulation.

- **Functional Scope** – the Service Domain designs have well defined procedures to specify the functions that are performed directly by the Service Domain and those that are to be supported elsewhere and accessed through delegated service calls (externalized). The discrete non-overlapping properties of the Service Domains provides a comprehensive and robust framework for defining the required internal/contained and external functionality. As noted the internal functionality needs to support the full life-cycle of the control record. Any function, information and action that does not have some aspect of the control record as its subject should be externalized.

In conventional process oriented design, the definition of utilities and other shared resource access is determined primarily by implementation considerations and feasibility – there is no high-level design partitioning discipline that enforces the correct scope of any particular functional application 'module'

2.2 Business to Technical Architecture – Mapping Service Domains

The BIAN SOA defines discrete business functional capacity partitions as Service Domains. The Service Domains are usually considered to operate as service centers – operational functions that provide (and consume) business services from other operational functions.

At the business architecture level, the Service Domains can be used as the elemental blocks for building different views of the business enterprise that are then used for different types of planning and analyses. This use of Service Domains is addressed in Sections 3 & 4 of this guide. This section looks at relating the Service Domains to the underlying systems architecture model views that can be used to help design the supporting business applications.

Business Capability Partition Vs Business Capability

A BIAN Service Domain is most accurately referred to as a business capability partition or business capability building block. To avoid any confusion, we use the rather unwieldy term business functional capacity partition in these guides. There is a subtle distinction between the capability partition represented by a Service Domain and an aspect of a business that is conventionally referred to as a 'business capability'. The Service Domain represents a discrete and generic business function or the capacity to perform some action such as *maintain reference details about a customer relationship* or *operate a network*.

A formal definition of a 'business capability' goes further to describe something that the business wishes to be able to do with assignable accountability and for which some associated value and/or motivation can be ascribed. The business capability combines the capacity to perform within ***specific organizational business context***.

The function performed by a Service Domain may be leveraged/reused to support different business capabilities with different associated business contexts and associated values and/or purposes. For example, BIAN has defined a Service Domain that tracks/determines a bank's credit view for a customer (Customer Credit Rating). Consider when this is involved in two different business capabilities:

1. (The capability to) Match products to customers
2. (The capability to) Negotiate product pricing with customers

The two business capabilities would both likely reference Customer Credit Rating. But the value/impact of the bank having an inaccurate credit perspective of the customer varies between the two. If say the credit perspective is overly generous the impact on product matching could be to recommend the wrong product, leading to a missed sale or the sale of an inappropriate product. The impact on the pricing business capability could be to offer too generous terms - a different value measurement.

Having the business capability view allows this context-based distinction to be maintained. BIAN is currently developing a business capability model to augment the current Service Landscape. The high level capability model is included in the latest release.

Mapping Service Domains to Business Applications

The Service Domains define partitions of the application logic and information/data that need to be reflected in the solution’s technical architectures. The way the Service Domains map to a technical architecture will vary for different technical environments broadly reflecting different ‘levels’ of sophistication in the service enablement. The mapping in three different technical environments is addressed in the next subsection. Before considering this mapping some more general statements are needed as to how the logical partitions defined by Service Domains line up with the business applications/systems in general. The terms used and descriptions of the different mapping arrangements is described in more detail in the How To Guide – Design Principles & Techniques. Those descriptions have been summarized here.

A stand-alone business application will have functionality that is typically represented by a collection of several Service Domains. It is also possible for Service Domains that combine many different tasks (such as product design or financial modeling) such that their implementation could require multiple (small or highly specialized) business applications. Sometimes a Service Domain will map neatly to a single business application. The most common situation however is where a business application has functional scope covering multiple Service Domains.

The diagram below captures these different Service Domain to business application mapping arrangements. It is used to explain the service operation support considerations when the mapping is not a convenient one to one.

- **Many To One** – when multiple business applications support the scope of a single Service Domain the issue is the support for service operations that rely on information or functionality that spans the business applications – where is the necessary consolidation of activity performed.

- **One to Many** – when a single business application covers the role of multiple Service Domains the issue is whether all of the service operations of the constituent Service Domains can be accessed externally (functionality can often be embedded/integrated in a way that compromises its ability to act as a discrete service center).

These service support issues are highlighted in the next Figure:

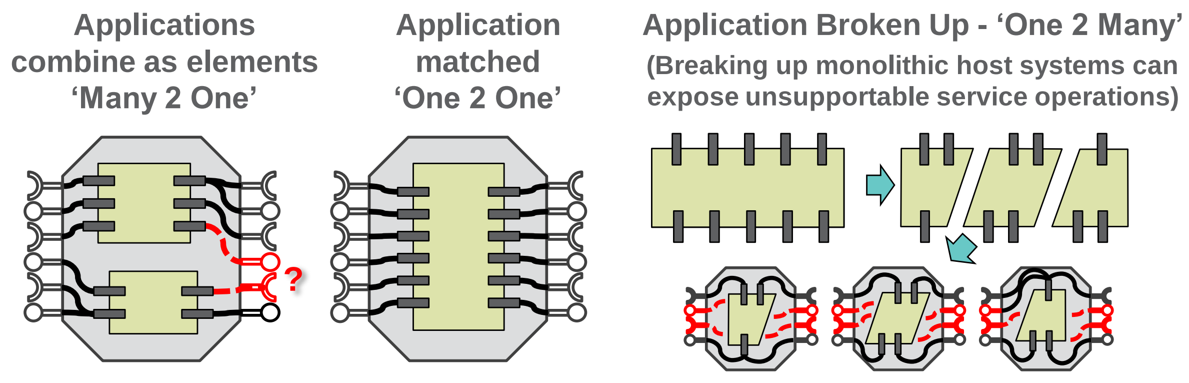

Applications combine as elements 'Many 2 One'
?
Application matched 'One 2 One'
Application Broken Up - 'One 2 Many'
(Breaking up monolithic host systems can expose unsupportable service operations)

Figure 3: Mapping Business Applications to Service Domains

In all of the mapping options described the service boundary of the Service Domain and the business application are 'aligned' meaning a business application is fully contained within the scope of a Service Domain or a Service Domain is fully contained within the boundary of a business application. The case when they are not aligned is when the same service operations for a Service Domain somehow straddle two or more business applications. In this case there will need to be duplicated/redundant logic in more than one business application and the discrete/non-overlapping principle behind the BIAN service based design will have been compromised.

Business Architecture Vs Systems Architecture views of a Service Domain

The mapping arrangements described so far assume that the business application performs a discrete business role (and can therefore be mapped uniquely to one or more Service Domains). When considering the scope/mapping of application logic there are two situations where the relationship between the logic supported by the software components and the discrete business functions of the enterprise is not directly and uniquely resolvable. This is the case in two main situations:

1. The application module is a 'utility' function that can be used in many different contexts. Each instance of use is completely independent/unaware of other instances. For example, a 'library' of complex algorithms could be coded and reused in many different applications supporting many different Service Domains.

2. The application module provides a 'common solution' that can be configured to support the needs of different business functions. An example would be in the area of product fulfillment. There could be a collection of products such as different types of loan that are captured as discrete business entities at the business architecture level (and so would have different Service Domains). But in operation they have very similar behaviors such that an application solution built for one could be reconfigured and redeployed to support the others. As with the utility function, each application deployment is functionally independent/unaware of other deployment instances.

This mapping of utility and common solution application modules to Service Domains is shown schematically in the next figure:

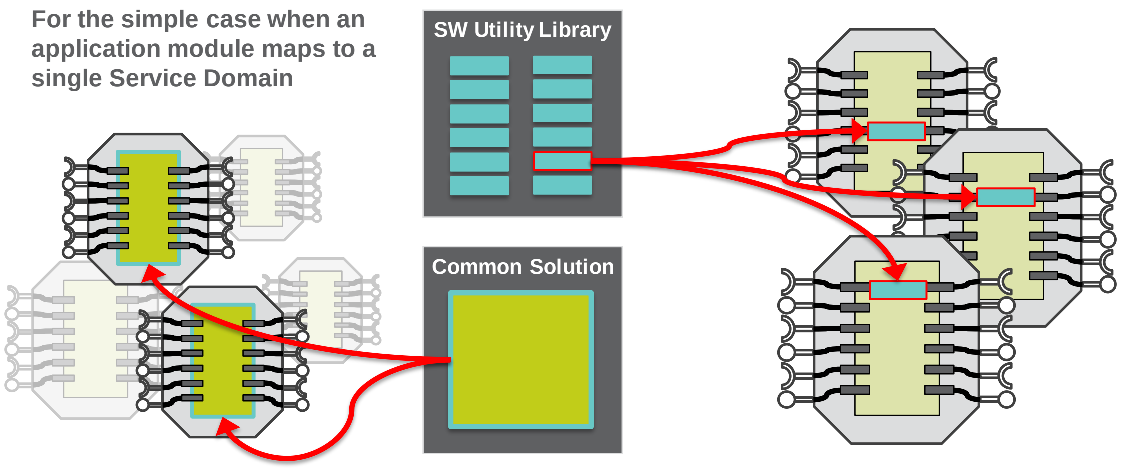

For the simple case when an application module maps to a single Service Domain
SW Utility Library
Common Solution

Figure 4: Aligning utility and common solution application modules to service domains

The use of shared utility and common solution application modules is an important aspect of effective software development and deployment. The use of these kinds of application module can be properly represented at the system architecture level. It is however not an aspect of the business architecture representation because the business architecture level intentionally shows only discrete business activities. These business activities may be supported by any appropriate combination of application modules including unique logic, re-used utility elements or employing a configured instance of a common solution.

The tracing of utility solution elements and the possible scope of common/shared solutions can be overlain on the business architecture representation. Where there is a common pattern to this the mapping can be a useful guide for application development. The Figure below shows how utility and shared solution options might be related to a BIAN business architecture model.

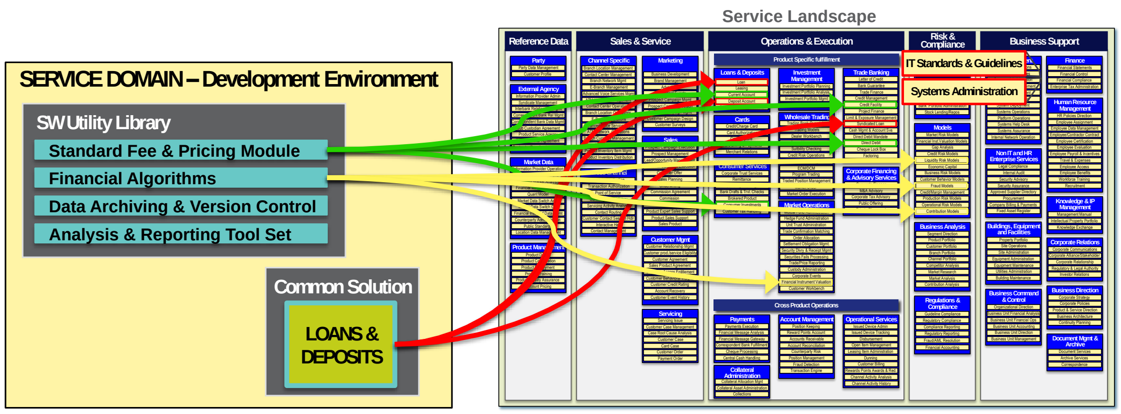


Figure 5: Service Landscape with shared and common solution overlain

Vendor Agnostic Application Model

BIAN has established a Working Group to explore the topic of mapping BIAN designs to more detail application architectures in more detail. The goal of this Working Group is to define and develop a ‘vendor agnostic’ application architecture view of the BIAN Service Landscape.

In addition to the broad alignment to the BIAN Service Domains outlined above this group considers how application logic may need to be partitioned to deal with performance and security considerations. It also considers how to represent application logic that is not reflected in the BIAN model such as operating systems, operational/functional utilities and platform capabilities. The results of this Working Group will be included in later releases of this guide.

Service Domains can be mapped to Micro-services

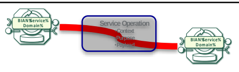
Micro-service architecture has a lot in common with the core design principles employed by BIAN. The Gartner definition of a Micro-service underscores this:

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

Micro-services can be defined at varying levels of detail. 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 functional scope 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.

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
The hierarchy used to build up solutions from elementary components	***Defines busines capability partitions as discrete and bounded (static) functionality***	***Defines the exchange of information and actions to support (dynamic) business behaviors***
Elemental Component	BIAN Service Domain = Micro-service 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)*	 BIAN Service Domain Service Operation •Context •Purpose •Payload BIAN Service Domain 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)*
Bundles – initial thinking is that bundles need to be support at two levels: 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 6: BIAN Service Domains related to micro-services

2.3 Service Domain Clusters

A Service Domain Cluster describes a grouping of related Service Domains. A 'Cluster' could be used to define a grouping that correspond to an organizational 'segment' as defined by TOGAF such as a business unit, profit center, division or enterprise. It can also be used to define a grouping that maps to the functional scope of a business application or production system. The second type of grouping is considered in more detail here.

The mapping options just described relates BIAN Service Domains to discrete conceptual application partitions recognizing that there is not always a simple one-to one association of the functionality. Service Domain business application clustering takes this mapping one step further by taking into account considerations when the logical/conceptual design has to be translated into a physical implementation design.

The BIAN Service Domains each represent a discrete, non-overlapping business functional capacity. In theory (and in some technical environments) each Service Domain could be implemented as a stand-alone application and all business activity could be supported by service collaborations between these distinct applications. In practice the significant majority of business applications combine the workings of several Service Domains as an integrated business solution. The reasons for integrating functional partitions together include performance, operational coherence and integration considerations.

The various technical and commercial packaging reasons for combining functional partitions into an integrated application are not directly considered in this document. However, for a business application cluster of Service Domains it is necessary to define 'roles' that define how the individual Service Domain designs relate to the broader application portfolio. The Service Domains tend to play one of three roles in the context of the overall enterprise' systems portfolio as defined below:

Service Domain roles within a business application cluster are:

- ***Core*** – The Service Domain exists only in the business application represented by the cluster. 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). The Service Domain Current Account Mortgage Fulfillment would be a core Service Domain in the Current Account Mortgage Processing Application cluster...

- ***Proxy*** - Represents a capability that is likely to be repeated in other clusters and is included in the cluster to provide a local 'view'. 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 it needs to synchronize with the master instance elsewhere through suitable 'background' services. SD Party Data Management could be a slave proxy service domain in the Current Account Mortgage Processing Application cluster.

- ***Utility*** – As with the ‘proxy’ Service Domain role, the cluster contains a non-unique instance. But in this case 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. Position Keeping (the transaction journal) is a utility instance in the Current Account Mortgage Processing Application cluster

When Service Domains are grouped into a cluster the external boundary of the cluster can be defined by referencing the available service operation connections between any of the Service Domains within the cluster and the surrounding Service Domains with which they interact. For Proxy Service Domains additional external connections are needed to ensure their synchronization with other copies of the Service Domain maintained elsewhere.

An example of a business application cluster is shown below (note only a sample of service operation connections and surrounding/referenced Service Domains is included for simplicity):

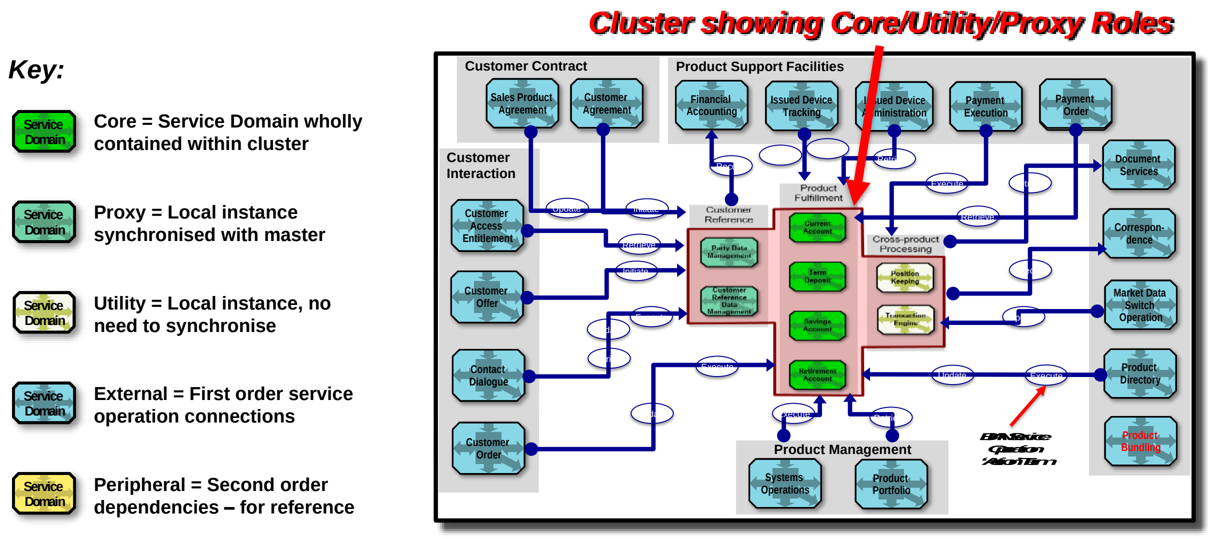


Figure 7: Core banking Business Application Cluster

2.4 Mapping Implementation Level Functionality to a Service Domain

The BIAN Service Landscape provides high-level descriptions of the BIAN Service Domains and their service operation exchanges. Beyond the formal content of the standard, the Business Scenarios and wireframes also provide examples of how the BIAN Service Domains may collaborate in different situations and in time BIAN may develop and provide other example views to assist with the adoption of the standard.

The content outlines the Service Domain’s mainstream business operational features and exchanges at a high level. The intent is to define clear functional partitions/boundaries in a way that is implementation independent and unambiguous.

The descriptions should be interpretable into any prevailing technical environment and they should be sufficiently detailed for the Service Domain design partitions to be consistently interpreted between different deployments.

The Service Domain boundaries can then be used to align and arrange application logic into discrete (non-overlapping) functional partitions with clear interfacing requirements that are well suited to service enablement in a SOA. The high level semantic BIAN definitions need to be extended to provide the necessary software implementation level detail.

In the previous release cycle BIAN added an additional level of detail to the Service Domain and service operation specifications. More recently BIAN has started to define its own business object model (BOM) to specify the service operation content. The BIAN BOM is based on and extends the industry standard ISO20022 Business Model as mentioned earlier in this guide.

This additional level of design detail for the Service Domain is based on breaking down the main behavior of a Service Domain, that defined by its ‘functional pattern’, into finer grained behaviors called ‘behavior qualifiers’. The behavior qualifiers for a Service Domain are used to add detail to the internal working, the business information governed and the purpose and content of the service operations the Service Domain offers and consumes. Different behavior qualifier types are defined for each of the BIAN functional patterns as described in more detail in the BIAN How to Guide – Design Principles and Techniques.

In this section three main ways the BIAN specification content can be extended are described in more general terms:

- **Service Domain functionality** – BIAN does not define the internal functioning of a Service Domain in any great detail but the functional scope can be inferred from the business role/purpose, control record and service boundary. This outline functional description can be extended using functional and non-functional checklists
- **Service Operation** – the BIAN service operations provide a semantic description of the exchange dependency between two collaborating Service Domains. This definition can be extended in two key ways – 1) the information content can be defined in more detail by mapping to underlying message exchanges; and 2) the protocol or orchestration of the interaction can be defined in terms of the structure/choreography of the dialogue.
- **Semantic APIs** – the BIAN Service Domains and service operations can be used as a high level for defining standard application programming interfaces (APIs). A specific BIAN How To Guide is available on this subject. It’s main content is summarized later in this guide for ease of reference

A fourth way the Service Domain specification is being extended is through the definition of the business information (and associated data representation) governed and referenced by the Service Domain and its service operations. BIAN is developing its own business object model (BOM) that builds on the industry standard ISO20022 Business Model.

BIAN is extending the model as needed to map to the specific BIAN Service Domain and service operation structures and to add content where it is missing. BIAN is working in close collaboration with ISO, submitting definitions of the extensions with the intent to maintain alignment. It is hoped that in time a single integrated conceptual object model will evolve. To this end as extended definitions of the Service Domains are created as part of BIAN's Semantic API initiative the business information profile is mapped to the BIAN BOM/ISO20022 model and the extensions registered with ISO for consideration.

Possible Service Domain functional specializations

When interpreting the high level BIAN designs there will often be a need to add or make amendments to handle site-specific variations before additional detail is mapped to the structures. These variations may be required to deal with considerations such as local geo-political constraints, aligning with legacy systems behaviors, supporting unique differentiating business practices and/or technical environment implementation features. Whatever the reason for these specializations, as long as the core role and purpose of the individual Service Domains remains intact, the anticipated benefits of the BIAN SOA standard will be realized.

The key mechanism that can be used to ensure the core role/purpose is retained as the Service Domain is specialized by adding implementation level specification detail (and optionally local specializations) to the Service Domain and its service operation specifications is the control record. As mentioned in the discussion of 'externalization' earlier in this guide, all service operation fulfillment, internal functional features and associated business information use needs to be relatable directly to the control record.

In the case of service operations and the linked messages underlying the requested action and the information content needs to pertain to the definition of a control record instance and if appropriate initiate some action that relates to its life cycle behavior. Any extensions to the business information definitions and associated data structures should also relate to the structure and content of the control record without changing its basic scope or definition.

2.4.1 Extending the functional definition of the Service Domain

The BIAN definition of a Service Domain considers the internal functionality largely to be a 'black box' – BIAN does not attempt to specify any internal working patterns or architectural structures in any great detail. BIAN merely clarifies at a high level what business functionality it should contain in order to fulfill its business purpose generally, to support its offered service operations and to outline what business functionality it may need to access to through delegated service operation calls to other Service Domains.

The main reason BIAN does not expand on the Service Domain functionality as part of the canonical standard is that BIAN’s focus is to help improve interoperability between business functional capacities and not the effectiveness of those functional capacities themselves. As a result the standard only seeks to define formally the service exchanges that connect the business partitions. For this it is only necessary to outline the purpose/role of a functional partition in order to be able to explain/match its offered and consumed services.

Though a limited definition of the Service Domain functionality is sufficient to specify its service operation use, it has been found that more detailed functional descriptions are very useful to implementation teams using the standard. The improved descriptions are needed to ensure that the teams correctly interpret the Service Domain functional partitions, particularly when relating the Service Domains to existing systems and/or development projects. But as the internal workings of the Service Domain can change and evolve, any more detailed functional descriptions are not canonical. Instead they only provide some prevailing examples of mainstream functionality as a guide.

The limited functionality description provided for the BIAN Service Domain can be easily expanded upon using the simple mechanism of a ‘checklist’. The checklist provides a simple structured framework to list the prevailing functional and non-functional properties that might be expected to be in place for a Service Domain (or more precisely the business applications supporting the activity scoped out by the Service Domain). The checklist includes the main prevailing features and can optionally include sub-structures to list more specializations features aligned to requirements such as:

- ***geopolitical requirements*** – specific traditions and laws/regulations,
- ***advanced levels of sophistication*** – advanced practices yet to become standard
- ***scale/segment*** – different properties that might apply to types of financial institution or specifically to large enterprises

An example of a basic feature checklist table is shown in Section 2.5 of this guide.

BIAN does not currently maintain feature tables for the Service Domains. Like Business Scenarios the feature tables are not canonical and only provide example content. Furthermore, the functional feature lists can be expected to change as new practices emerge. This guide only describes the structure and use of feature tables as a tool. It is anticipated that banks and solution providers will develop and maintain their own feature tables or equivalent as might be necessary. BIAN may consolidate and make available example feature tables for Service Domains if this is found to be useful in the future.

When developing the feature lists for a Service Domain the same externalization tests already described for specialization should be applied to the content. Essentially all listed functionality should be directly relatable to instances of the Service Domain control record and its particular life-cycle behaviors.

When considering the fit of a functional feature to a Service Domain it can help to consider the Service Domain in the context of one or more Business Scenarios. It can be easier to confirm the decision to externalize a function (that does not relate to the control record of the considered Service Domain) if the correct location for the functional feature can be assigned to some other Service Domain.

With the addition of behavior qualifier types to the specification of a Service Domain there may be corresponding refinements that can be made to the structure of the feature checklist table to reflect the different behavior qualifiers for a Service Domain. This option may be explored in later versions of this guide.

2.4.2 Mapping service operations to messages

BIAN service operations describe a high-level dependency between two Service Domains. They list the exchanged business information and may refer to services/actions that are requested. The BIAN service operation does not define the protocol or choreography of the interaction as this is typically implementation dependent.

In an earlier release BIAN defined a comprehensive checklist of the types of information that might be maintained by a Service Domain and that could be referenced in the payload of its offered service operations.

BIAN used filtering based on the Service Domain’s particular functional pattern and the action term of its default service operations to define candidate service operation content. From the previous release onwards BIAN is progressively replacing this checklist-based information content definition approach. The new approach defines specific semantic information content for individual Service Domains and their service operations. This activity is closely coordinated with the BIAN Semantic API initiative.

The two content definition approaches and the structure of the information in a BIAN service operation definition are described in more detail in the next section. This section first describes the steps of a procedure that maps service operations to underlying machine level message standards when they are available.

As noted earlier the service operation defines the information exchanged. It does not define the protocol or choreography of the exchange as this is implementation specific. For example, a service operation exchange could be realized by a simple two-way ‘handshake’ of information or could result in a complex iterative exchange of underlying messages. In this context a message defines the data content exchanged in appropriate detail.

A service exchange may involve some combination of:

- The movement or assignment of some facility or resource
- A free-form person to person dialogue/negotiation
- ‘Structured’ and unstructured information exchange person to machine and machine to machine

As BIAN's focus is on improving application to application interoperability the focus of the service operation definition is on the specific content related to the exchange of structured and unstructured information. Given the ever increasing ability of (AI) technology to infer structure from different information sources the boundary between structured and unstructured information is constantly moving with machines better able to interpret less structured information.

The term 'message' refers to standard data structures defined to support specific application to application exchanges. A message may include a combination of individual data items, structured data records and unstructured data. Standard messages have been published by a number of standards bodies. Of particular relevance to BIAN is the ISO20022 financial services message specification. Standard messages are key for several aspects of banking (payment in particular). Though published industry standard message specifications are only available for a small subset of the business activities covered by the BIAN Service Landscape at this time.

The precise structure of the BIAN service operation, in terms of the different fields, naming conventions, standard content and content explanations are more completely documented in the How-to Guide – Developing Content. These finer details are not so important here where a general process for matching up the service operation with the underlying standard message is outlined.

BIAN has undertaken a number of initiatives over recent years to explore repeatable ways to map service operations to messages. A general approach is described below. It has been derived in part from research initiatives performed by students at Carnegie Melon University in collaboration with BIAN and PNC Bank. The final reports for these studies are available at BIAN.org.

The mapping approach uses Service Domain design elements and their service operations. These are explained in the How-to Guide – Design Principles & Techniques and summarized here for ease of reference. The key BIAN design elements/considerations include:

- **Service Domain 'functional pattern'** – every BIAN Service Domain has a standard operational behavior (its functional pattern). It performs this function on instances of a selected type of asset. It is responsible for fulfilling its function for the complete life-cycle (from start to finish) for each instance.

  For example, the Service Domain 'Product Design' has the functional pattern 'DESIGN' on the asset type 'product/service', (here the term 'product/service' refers to the capacity to support some product/service and the 'asset' is the intellectual property of its specification).

  The full life-cycle for the instance of a product design spans the initial identification/registration of the design specification, through all specification/update cycles and usage scenarios through to the final termination/archiving of the design.

- **functional pattern** – BIAN has identified a number (18) of generic commercial behaviors that are applied to different asset types in the execution of business. For example, for an asset such as an ATM network there are several applicable functional patterns that represent the things done to maintain and leverage this resource for commercial advantage. These include managing/configuring, operating, maintaining and analyzing the performance of the ATM network. As noted each Service Domain's behavior is characterized by one functional pattern

- **asset type** – BIAN has used a simple hierarchical decomposition technique in order to identify the full range of tangible and intangible assets that may be found in any Bank. BIAN has also refined techniques to determine the correct level of granularity to perform this type decomposition in order to identify Service Domains that are elemental in their role. This technique is fully defined in the How To Guide – Design Principles & Techniques. As already noted each Service Domain's behavior is the combination of its functional pattern applied to the full life cycle 'processing' of instances of its specific asset type

- **generic artifact & control record** – As functional patterns describe a behavior they typically take the verb form. The generic artifact for a functional pattern simply describes some form of tangible record or document that can be associated with the execution of the functional pattern. For example the functional pattern 'agree terms' that describes the action of defining and maintaining governing terms has the associated generic artifact of an 'agreement'.

  A Service Domain applies one pattern of behavior (functional pattern) to one asset type. Its control record combines the functional pattern's generic artifact with the asset type. The control record can be thought of as a mechanism used to track/manage the execution of one occurrence of the Service Domain performing its business role for a complete life-cycle. For example, the Service Domain *Product Design* the functional pattern is design and its generic artifact is 'specification'. The asset type is 'product/service' (short for the capacity to deliver a product or service) resulting in a control record that is 'product/service specification'.

- **action terms** – the primary purpose for each service operation call is reflected in its action term. BIAN has identified a standard set of action terms to select from and each service operation uses one of these action terms. In general each action term defines the kind of operation that the service operation results in on one or more control record instances, for example activating, updating, requesting or retrieving (reporting) on that instance.

- **service operations** – a structured framework/template is used to capture the properties, naming and payload/content of a service operation. Note, that with the recent introduction of the behavior qualifier type and Service Domain specific behavior qualifier definitions, the service operations and their information content can be defined to a finer level of detail. This additional detail should improve the mapping accuracy, but the overall approach to mapping has not been changed

The steps in the general approach reference these design features in order to match messages to service operations. The general steps are described as applied to an individual service operation:

1. **Step 1 – Asset Type to object** – the service operation's host Service Domain's control record includes the asset type that is acted upon. This asset type can be mapped to the object or data type that is the subject of messages from the target message set. For example, the asset type could be a customer relationship and the associated object is the customer object. The selected messages will contain customer related data

2. **Step 2 – Functional Pattern filtering** – the BIAN functional pattern defines a constrained used of the asset type. This can be used to narrow the scope of the data related to the mapped object and this in turn can be used to filter/eliminate the mapped messages. Continuing with the customer relationship/customer object match, if the functional pattern is AGREE TERMS, the customer related data can be limited to that directly associated with the details that make up a customer agreement and any message not containing this type of data can be eliminated from further consideration

3. **Step 3 – Action Term alignment** – the action term provides a fairly precise definition of the purpose for the service operation call (the intended action to be performed). Many messages are similarly associated with some kind of intended use/purpose – mapping the action term to this when available can be used to further filter/eliminate candidate messages. As already mentioned, with the recent addition of behavior qualifiers to the BIAN model, this mapping may be done at a finer level of detail. This is where behavior qualifiers are used to specialize the purpose and information content of a service operation

4. **Step 4 – Service Operation payload** – the final step uses the semantic description of the business information content of the input and output parameters of the service operation. The content is mapped against the information payload of any candidate messages. This is done to confirm that the message contains all key information and may also can highlight redundant/excessive data content in the message for the intended purpose of the service operation. In the latter case a design decision is required as to whether the excessive content eliminates the message from the mapping

As noted, the selection and filtering of the messages described above does not take into account any message exchange 'choreography' that may be involved in the service operation exchange. The BIAN service operation simply defines the main information exchange dependency.

2.4.3 Semantic APIs

Semantic application programming interface (API) refers to using the BIAN Service Domains and their service operation interactions as a ‘top-down’ framework for defining standard aspects of an application to application interface. In the prior release BIAN included an update that added necessary detail to the Service Domain and service operation specification approach to support the use of BIAN to define APIs. In this latest release a selection of Service Domains has been specified following this more detailed approach. In parallel BIAN has developed the BIAN API Exchange – an open-source development ‘sandpit’ where the service operations of 67 Service Domains have been translated into RESTful API endpoint specification (over 900 in all). BIAN will continue to develop extended Service Domain specifications and expand the coverage of the exchange as fast as is practical.

The API approach and an outline of how the BIAN standard can be used is covered later in this section of the guide. A complete description of the BIAN API initiative can be found in the BIAN Semantic API How To Guide that is published with the latest Service Landscape release.

All BIAN How to Guides are pitched for business and application architects. API guidance that is targeted at solution developers is partly integrated into the API Exchange. Additional guidance and possibly a separate ‘practitioner’ guide will be defined based on feedback from users.

2.5 Applying BIAN in different technical architectures

The BIAN model defines the business functional capacity building blocks as discrete partitions that are suited to service enablement. Though it can be highly beneficial to relate the high level BIAN Service Domains to a service oriented systems architecture (SOA) this is not mandatory. Here we describe three ‘types’ of target technical architecture to describe the progression towards a ‘pure’ service oriented architecture:

1. **Type 1 - Conventional (legacy/core) system rationalization** – in this example the BIAN Service Doman designs are used to assess an existing application portfolio. The Service Domain partitions are used to identify duplication and fragmentation of the business logic and information between the business applications

2. **Type 2 - Host renewal/ESB integration and application/system assembly** – building forward from existing system rationalization and synchronization, technologies such as an enterprise service bus (ESB) can be used to develop shared service capabilities and reduce redundancy across the application portfolio

3. **Type 3 - Loose coupled distributed/Cloud systems** – the most advanced use of technology considered is that of the highly distributed internet and cloud environments, where solutions are loose coupled and fully service enabled. This approach also fits with the adoption of rapidly emerging micro-service architectures

The BIAN Service Domains and their service operations collectively represent a complete, organized and non-overlapping description of all of the functional building blocks needed to assemble any banking business application. The systems support for the Service Domain building blocks and their interactions can be realized in different ways. If the business applications are aligned to the Service Domains effectively then the operational flexibility and efficiencies of a SOA can be realized to varying degrees depending on the technical environment.

Before describing how the BIAN designs are interpreted in different technical environments, it is necessary to make a distinction between two aspects of business operation that are captured in a Service Domain's specification as these aspects will be interpreted differently. To date, a Service Domain has been described as a business functional capacity partition that performs a business role and that is engaged through its offered service operations and may subscribe to services from other Service Domains as needed. This behavior is suitable to describe a service based implementation but the same business functionality may also be implemented in a less flexible 'hard wired' technical environment where the connections are point to point interfaces rather than being realized being through some flexible service based mechanism.

The Service Domain can be divided into two components – its functional core and a 'service enabling' wrapper that handles the interactions with other Service Domains as shown in the Figure:

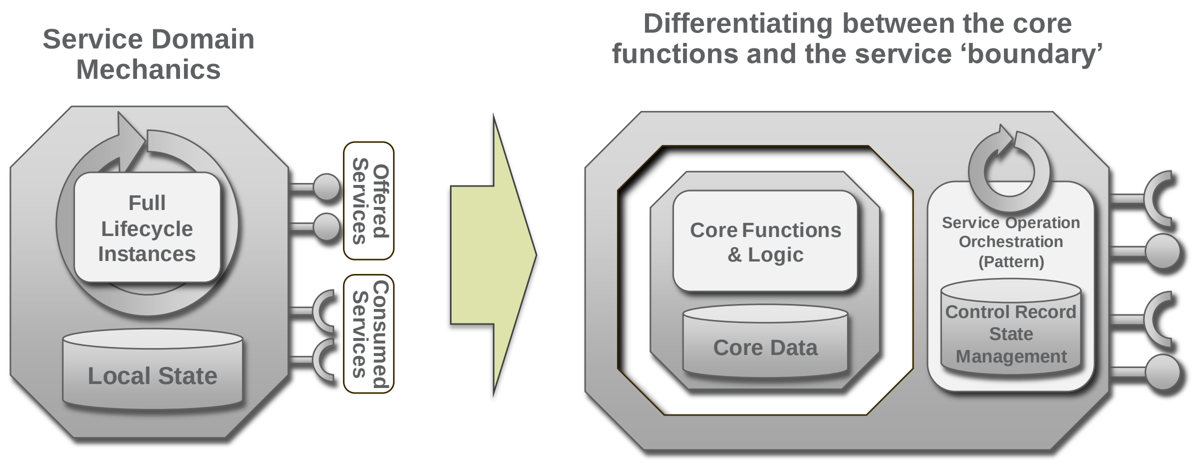

Service Domain Mechanics
Full Lifecycle Instances
Local State
Offered Services
Consumed Services
Differentiating between the core functions and the service 'boundary'
Core Functions & Logic
Core Data
Service Operation Orchestration (Pattern)
Control Record State Management

Figure 8: Service Domain broken into a functional core and service 'wrapper'

This distinction is referenced in the descriptions of the different technical implementation environments that follow.

2.5.1 Type 1 - Conventional (legacy/core) system rationalization

For legacy/core systems rationalization the Service Domains are used as a stable framework that defines non-overlapping functional partitions that can then be used to map the footprint of legacy/core applications to highlight different shortfalls. The Feature Checklists described earlier and the recent addition of behavior qualifiers can be used to provide a more detailed functional description of the Service Domains for mapping the existing application portfolio. Only the functional core of the Service Domain is used in the case, there is no assumption that any systems interfaces will be service enabled. The Service Domains are simply used to define the assessment framework.

As shown schematically to the left of the Figure below, most legacy business applications cover the scope of multiple but differing collections of Service Domains and so it is not meaningful to do a direct application to application comparison as two applications will typically have different functional coverage. Because the Service Domains do not overlap when the applications are mapped against them it is possible to do a like-for-like mapping by considering the application coverage for each Service Domain at a time and then consolidate the collection of assessments for all Service Domains in scope for an application in order to reach a determination as to its long term role.

This decision can become quite complicated as often a legacy system will not always divide up/modularize neatly along Service Domain boundaries. So if an application is found to be a good fit for some Service Domains and not for others it may not be possible to retain just the desired elements. The determination has to be performed on a case by case basis, but the Service Domain framework does at least give a clear indication of where an application has strengths and weaknesses to feed into that more objective selection assessment.

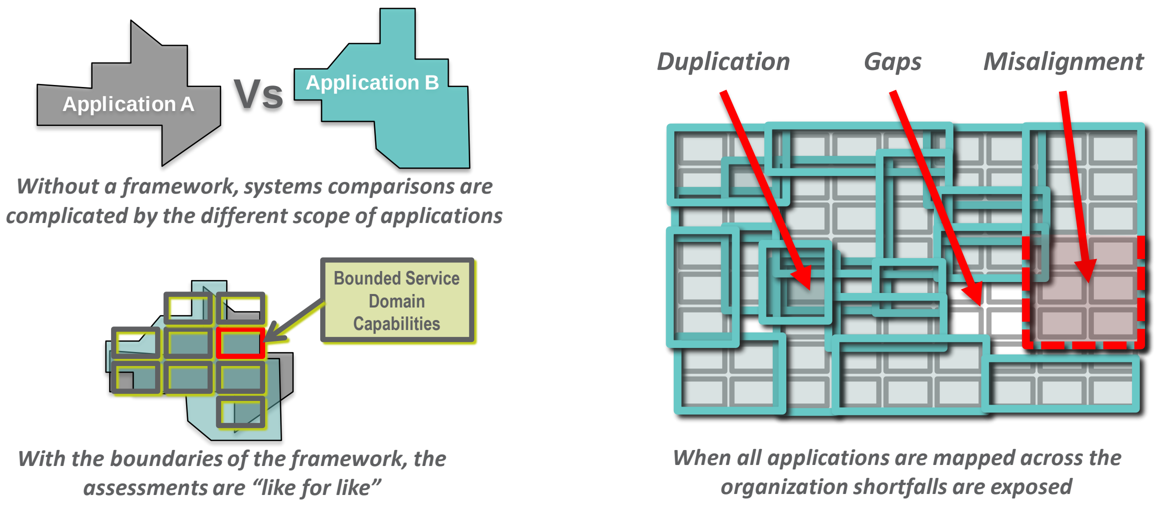

Application A
Vs
Application B
Without a framework, systems comparisons are complicated by the different scope of applications
Bounded Service Domain Capabilities
With the boundaries of the framework, the assessments are "like for like"
Duplication
Gaps
Misalignment
When all applications are mapped across the organization shortfalls are exposed

Figure 9: Using BIAN Service Domain partitions for comparisons

The schematic mapping on the right shows the Service Domains as the background grid and then overlays the functional footprint of the existing business applications. Three different shortfalls are highlighted;

- **Duplication** – perhaps the most obvious is where two or more business applications perform the role of the same Service Domain. As noted below this may or may not be an issue, but at this stage it highlights potential redundancy
- **Gaps** – the Service Domain feature checklist may include functional features that are not currently fully supported and these will show up as gaps in the mapping. It may also be possible to see which legacy applications are the best candidates to expand to cover these requirements
- **Misalignment** – this is a problem usually suffered by the better business applications. Built to support some particular business function they are subsequently extended into other areas as they offer the easiest/lowest cost solution. The problem can be that an application designed to support one function can become compromised when it tries to support many additional and potentially conflicting operational requirements

The mapping of the application portfolio can provide clear and concise insights into the overlaps/redundancy in the bank's application portfolio. Most banks suffer from significant levels of redundancy due to a history of siloed implementation and business acquisitions. Also often as systems are replaced the old systems are not always fully decommissioned.

The redundancy revealed by the mapping needs to be evaluated in more detail to determine the extent to which the duplication of functionality and business information leads to fragmentation and processing consistency issues. This analysis is intended to determine the extent to which the redundancy is causing a 'synchronization' problem or overhead across the business. This 'synchronization' issue applies differently for application logic and for information/data:

- **Duplicated Application Logic** – when two (or more) business applications perform the same business function using different application logic this can lead to inconsistencies in operational behavior. This impact can be minimal, or can expose the bank to significant additional costs, lost income and risk. Different factors to consider in the impact assessment include employee training and productivity, inconsistent customer experiences, operation inefficiencies/complexity, exposure to increased operational risk, inability to track, assess and report on activity consistently, increased overhead when making changes. There may also be operational synergies/streamlining opportunities that are compromised.

- **Duplicated Business Information/Data** – when two (or more) business applications maintain their own views of the same business information the synchronization issues are obvious. Changes or updates to information in one place may not be accurately reflected elsewhere leading to errors. Updates may also get lost or overwritten and the ability to consolidate, analyze and react to business information is compromised.

Looking at the issue of duplicated application logic from another perspective reveals that the extent and impact of the synchronization overhead just described (and hence the opportunity for improvement) may be far greater than first anticipated. In the Figure below an example has been developed for a notional stand-alone loan business application. The Service Domains accessed during the on boarding business scenario have been mapped into a simple ‘application architecture’ (9 Service Domains are involved in the origination scenario). Clearly many more scenarios could be considered to determine a comprehensive collection of Service Domains that would be included in the business application, but this sample is sufficient for the purposes of this exercise.

A determination has then been made for each Service Domain asking whether it is likely that this Service Domain will be duplicated in other business applications or is its business role unique to this particular business application? As can be seen to the right of the Figure eight of nine considered Service Domains are candidates for duplicated use.

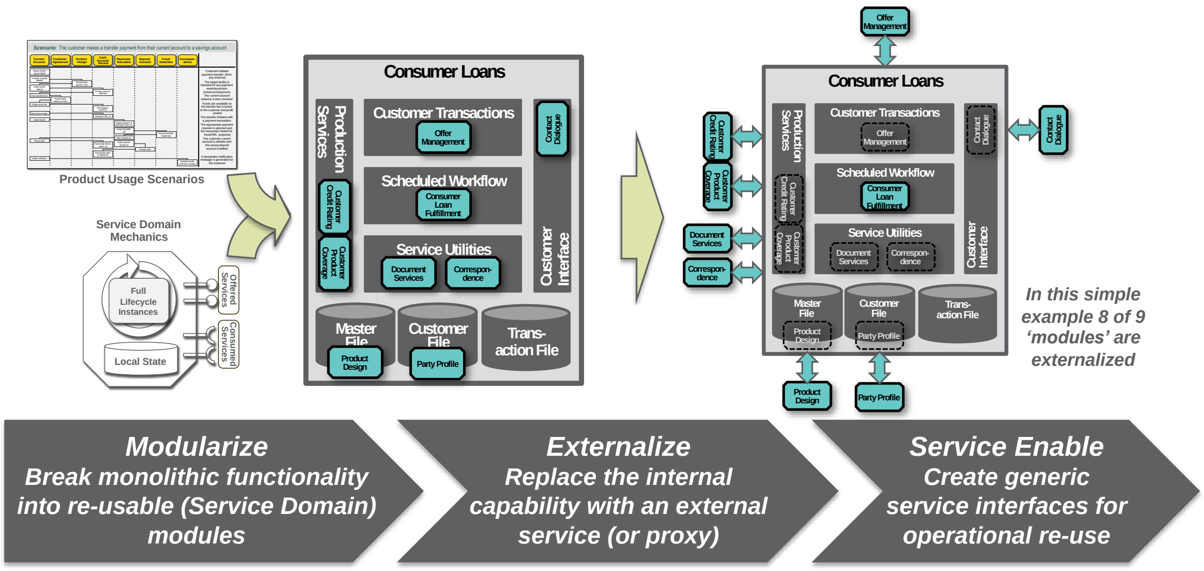


Figure 10: Externalizing Service Domains in an application

The example was not selected to demonstrate a disproportionately high level of redundancy – it is a pattern that is likely to be found for the significant majority of core/legacy business applications. For some application logic and/or business information the issue has long been recognized and shared solutions have been implemented to mitigate this. Obvious examples being customer information and relationship management, customer servicing and many shared back office functions.

Leveraging the discrete, non-overlapping properties of Service Domains to specify shared service capabilities can help reduce the synchronization overhead. It provides the many considerable benefits of operational re-use. This approach is closest to the Type 2 technical environment described next

2.5.2 Type 2 - Host renewal/ESB integration and application/system assembly

The second type of technical environment is where existing core/legacy host systems are service enabled with the specific intention of supporting shared access to function and data. This is done using service enabling technologies to provide access to their established host systems such as an enterprise service bus (ESB). Without any specific organizing blueprint to specify the scope and content of the enabled services the tendency has been to define fairly fine-grained services that provide access to utility functions and/or fairly narrow data sets often aligned to existing host access interfaces.

These fine-grained services can improve new business application development by providing re-usable software utilities. There are a number of limitations with this approach including:

- **Software re-use Vs business capability re-use** - The re-use of a software utility does not always drive through to the rationalization or re-use of operational business capabilities
- **Complex and unstructured service libraries** - Because the services are fine-grained this can lead to large and complicated collections of overlapping services that can be hard to categorize, maintain and reference
- **Proprietary services** – the defined services often include host application specific/proprietary features that can lock-in legacy systems and compromise the ease of assembling applications using services from different wrapped host sources
- **Synchronization** – the issue of traceability to function and data highlighted with the prior discussion of application portfolio rationalization persists if hosts systems are simply service enabled with no specific blueprint or design to help reduce any redundancy conflicts between host systems

The use of the Service Domain partitions as a framework to reveal duplication and the associated synchronization issues with overlapping business applications already described can be taken further with the definition of shared services. The Service Domains define non-overlapping functional partitions that cover all business activity. Accordingly, their associated service operations can define a comprehensive directory of non-overlapping business service operations.

When BIAN Service Domains and their service operations are used to define the service directory for the ESB then their implementation can be used to obscure and in time progressively support the elimination of the redundancy in the application portfolio and greatly reduce the application synchronization overheads. In this type of technical environment both the functional core and service wrapper components of the Service Domain specification are used in the solution specification.

The use of the BIAN Service Domains and their service operations to define the service directory for the ESB is shown schematically in the next Figure:

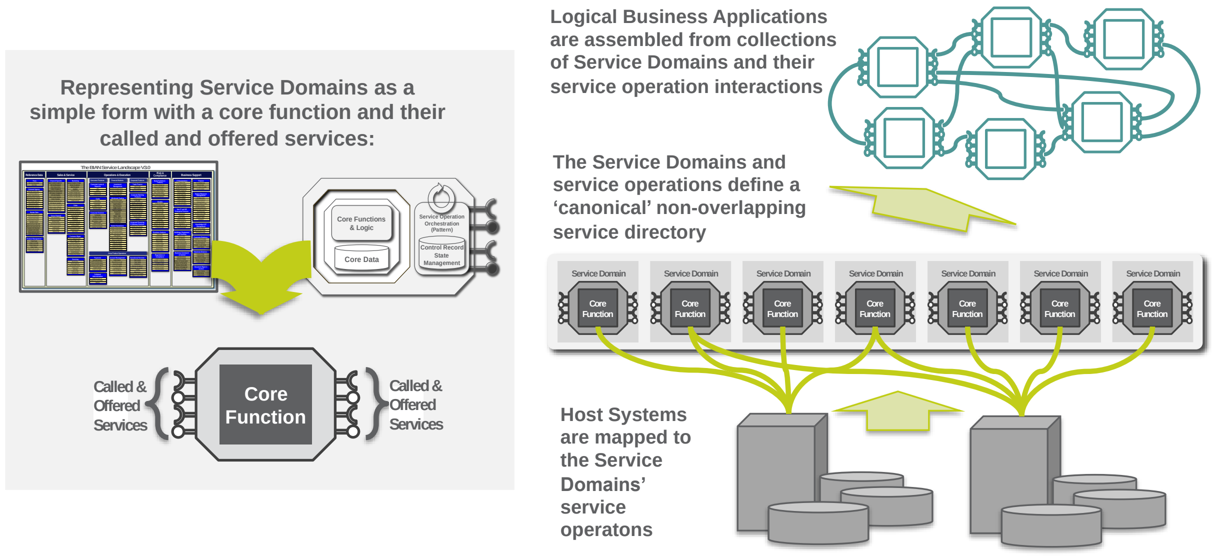

Representing Service Domains as a simple form with a core function and their called and offered services:
Core Functions & Logic
Core Data
Service Operation Orchestration (Pattern)
Control Record State Management
Called & Offered Services
Core Function
Called & Offered Services
Logical Business Applications are assembled from collections of Service Domains and their service operation interactions
The Service Domains and service operations define a 'canonical' non-overlapping service directory
Service Domain
Core Function
Host Systems are mapped to the Service Domains' service operatons

Figure 11: The use of BIAN Service Domains to define a service directory for the ESB

On the left of the Figure the BIAN Service Landscape and its collection of Service Domains defines the overall functional coverage and on the right the full collection of service operations is enabled through the enterprise service bus. The host systems are mapped to the ESB. The ESB enabled services can then be 'assembled' to support different business applications.

Host Alignment

Mapping hosts systems to the ESB is potentially a very complex undertaking. One advantage with the use of the BIAN Service Domains and service operations as the service directory is that the Service Domains are highly enduring and non-overlapping meaning that it is usually possible to implement ESB service incrementally and have the services adopted progressively across the overall application portfolio.

The selection, prioritization and scheduling of the migration towards ESB based solutions is an important and complex consideration for any bank. It will need to balance the feasibility, cost and risk of implementation with the anticipated business performance gains. The likely benefits in terms of improved operational integrity, flexibility and efficiency should however be significant in most cases.

For this discussion of mitigating considerations, it is assumed that the ESB services are mapped to existing hosts systems. It is of course possible that new business applications could be developed and deployed replacing existing host systems as part of the migration or external sources could be reference from the ESB also replacing existing services as necessary. These options do not change the approach significantly but are not considered here for brevity.

The BIAN service operations may combine both information access and requests to execute some type of activity. It is clearly important to map the information use and requested activity to the appropriate host capability and as noted in the previous section, there may be more than one candidate host system to consider. The mapping process is described first as the steps are applied a single host system:

1. ***Service Domain Alignment*** – the business role of the Service Domain is mapped to the host business application confirming the business purpose for the Service Domain is properly aligned to and is covered by the functionality of the host business application. Note that this association should be based on the functional purpose and not just the profile of business information. It is possible that different business requirements use similar business information. For example a Service Domain that supports some aspect of marketing may use similar customer information to a Service Domain that supports credit decisioning.

- ***Service Operation Alignment*** – once the role of the Service Domain alignment to the host business application is confirmed the next step is to consider the more targeted purpose of the specific service operation. The BIAN service operations are defined at a fairly generic/coarse grained level. It is possible in some cases that it is necessary to break the individual BIAN service operation into a collection of more specialized/finer grained service operations prior to mapping to the host facilities. With the latest releases the concept of 'behavior qualifiers' has been introduced. This can be used to define finer grained service operations that will assist with this mapping effort as more specifications are progressively developed across the Service Landscape.

  The purpose of the service operation can usually be inferred from its 'action term', (and where provided the more detailed behavioral qualifier), its general description and where applicable the service operation control parameters. It is necessary to consider the intention of the service operation in as much detail as possible from the semantic definition and confirm that the host access is well matched in terms of intent and underlying functionality.

- ***Business Information Alignment*** – the BIAN semantic service operation description will usually also contain a fairly comprehensive description of the likely business information exchanged. In some cases, the BIAN service operation may also be matched to a more detail message specification where an industry standard message has been defined. The semantic content and/or message definition is used to check that the information content available from the host system is sufficient for the purpose of the service call.

Once the service operation is matched to the host business application the service operation fulfillment has to be enabled through the ESB connection to the host business application. In the case where some type of action or response is required the mapping has to be made directly to some existing or newly developed host interface. This will often be realized through parameter-based access to the interface. If no suitable interface exists or there are functional shortfalls it may also be necessary to front-end the host with some form of compensating capability or enhancing the host system itself to fully support the service operation.

The host mapping may be complicated by several factors, such as partial matches or the sourced information may reside on multiple sources. As noted it may also be necessary to compensate for some shortfall in the host system data or logic. In addition to supporting the compensating logic needed to deal with functional and data shortfalls and coordinating multiple host access requirements there are several performance enhancements that can be built into the ESB capability including:

- ***multiple sign on*** – the ESB may be able to manage multiple host session sign on sessions to reduce the connection complexity
- ***master/slave reconciliation*** – the service alignment may help resolve duplication by identifying suitable master/slave definitions of overlapping host function and data. Facilities to synchronize between the host systems can be considered to operate in the background host environment
- ***advance data look-up and caching*** – more sophisticated traffic analysis can be used to implement advanced look-up (look ahead facilities anticipating related data sets and retrieving the data in advance to reduce access latency) and data caching where retrieved data that is likely to be needed again is kept in a suitable caching capability

The Figure shows how ESB service operations may be mapped across the data structures of multiple host business applications

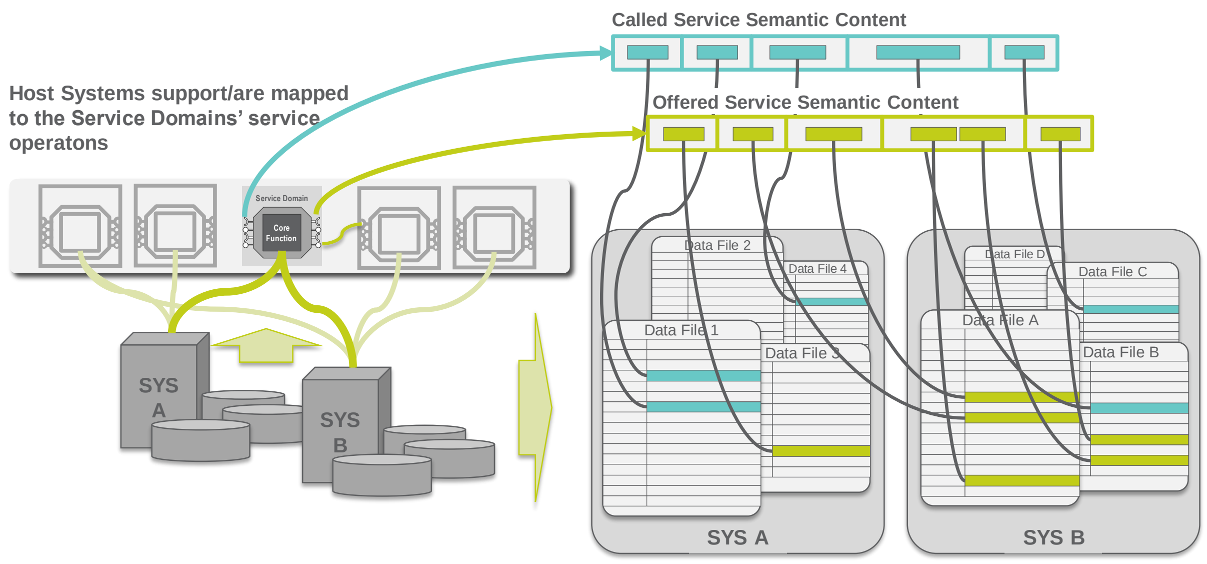

Called Service Semantic Content
Host Systems support/are mapped to the Service Domains' service operatons
Offered Service Semantic Content
Service Domain
Core Function
SYS A
SYS B
Data File 2
Data File 4
Data File 1
Data File 3
Data File D
Data File C
Data File A
Data File B
SYS A
SYS B

Figure 12: Mapping the ESB to host data structures

Application Assembly

The ESB will present available service operations that should provide integrated and coordinated support to the application portfolio. The services could simply be made available individually to existing end-user applications. Or they could support different application assembly environments and support more flexible application development.

The ESB mechanism can link to existing host systems and may provide access to other sources, for example external web based capabilities. An appropriate application assembly environment could support the assembly of business applications that combine wrapped host business applications (perhaps supporting transaction processing) with cloud based CRM capabilities

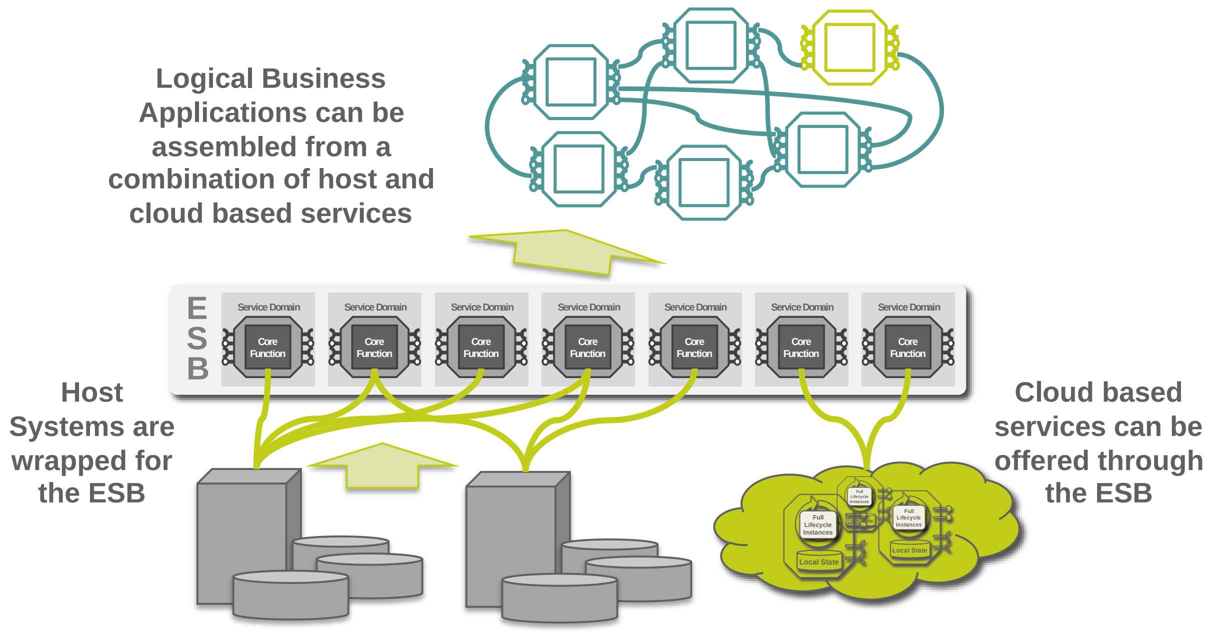

Logical Business Applications can be assembled from a combination of host and cloud based services
E S B
Service Domain
Core Function
Host Systems are wrapped for the ESB
Cloud based services can be offered through the ESB
Full Lifecycle Instances
Local State

Figure 13: ESB solutions integrating host and cloud based service solutions

The ESB approach can be used to migrate progressively towards a business application configuration where each operational service is offered by a single source and re-used whenever needed across the overall application portfolio. Such an arrangement eliminates operational duplication and maximizes business capability re-use.

2.5.3 Type 3 - Loose coupled distributed/cloud systems

The third type of technical environment is the highly flexible, distributed and connective networked platforms such as the Internet, the cloud (here 'cloud' includes private, public and hybrid configurations) and more recently micro-service architectures. This type of environment can be considered as a progression from the ESB environment to something that is a 'pure' service oriented architecture.

In this solution the mapped host systems that were presented through the structured ESB are replaced by freestanding business capability ‘containers’ that are made available to collaborate over the network as autonomous service centers. Some of the main operating characteristics of this ‘loose coupled’ highly distributed networked service environment include:

- *Functional capabilities operate independently* – each service center can act as a free-standing and independent functional ‘container’ with its own internal processing logic, data storage and state management. It runs to its own schedule/timing, calling services and responding to external events/triggers as it deems appropriate

- *Communications are through service calls* – the exchanges/interactions between the service centers are all by service operation calls. These exchanges will typically involve the exchange of structured data that can be formatted in one or more data messages

- *Varying levels of required information precision* – two collaborating service centers may only need to agree/coordinate the meaning of information elements at the more approximate semantic level. This reduces the requirement to adopt common data formats and structures for machine representable data. The topic is covered in more detail below

- *Exchanges use queue and event based mechanisms* – the networked service centers operate asynchronously. When one requests a service from another it can continue with other tasks and monitor for and act on the response when received. Operations should also be ‘defensive’ - dealing sensibly with delayed, missing or erroneous responses (and requests)

Business execution in highly distributed environments is typically event driven. Something triggers activity in one center that then need to call on services from other service centers. These ‘secondary’ service centers may then also call on other service centers before the can respond. The processing of the original trigger may result in a ‘cascade’ of many ‘nested’ levels of service interactions across the network until all necessary processing and responses have been completed and the network reaches a new stable state.

The BIAN Service Domains can be aligned one-to-one to the service center ‘containers’ just described. Both components of the Service Domain specification are used. The *functional core* defines the role of the container, outlining its required internal features and the *service wrapper* handles service enablement within the network (service/state management and directory/connection handling etc.). The service wrapper contains the necessary logic to ensure that the Service Domain is aware of all other Service Domains it may call on. It also implements the called and offered service operations, typically using some form of queue and event management. The next Figure captures the networked nature of the connections.

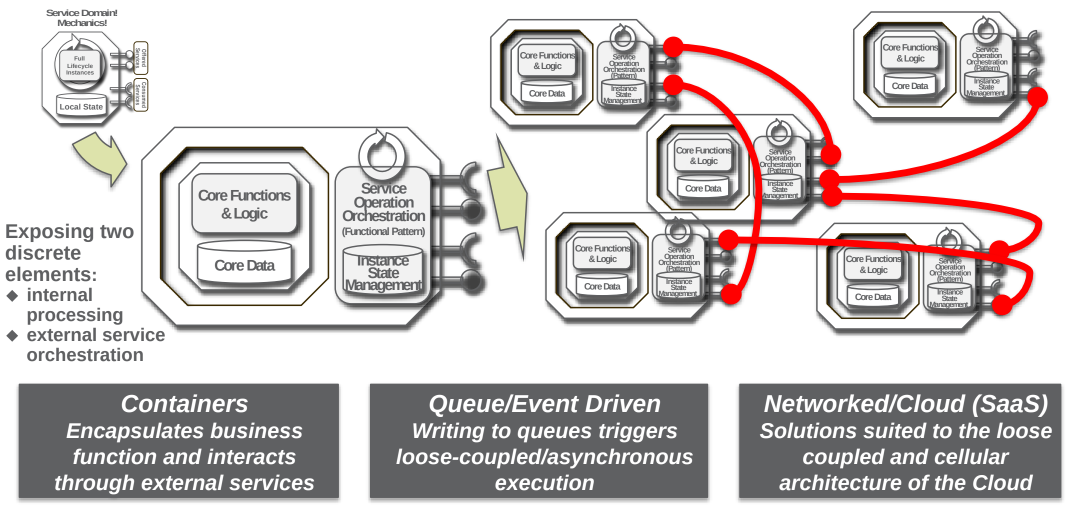

Service Domain! Mechanics!
Full Lifecycle Instances
Local State
Exposing two discrete elements:
◆ internal processing
◆ external service orchestration
Core Functions & Logic
Core Data
Service Operation Orchestration (Functional Pattern)
Instance State Management
Containers
Encapsulates business function and interacts through external services
Queue/Event Driven
Writing to queues triggers loose-coupled/asynchronous execution
Networked/Cloud (SaaS)
Solutions suited to the loose coupled and cellular architecture of the Cloud

Figure 14: Advance 'Cloud' technology solutions

Note that a service exchange as defined by BIAN may combine the physical assignment/movement of goods or resources, free-form dialogues between individuals as well as the more structured exchange of business information/data. The exchanges described in this guide focus on the exchange of structured and unstructured (machine interpretable) information. It is assumed that additional mechanisms will be made available as necessary to accommodate the other forms of traffic such as the physical distribution of goods.

Service Information Precision

As noted above service exchanges in the network require different levels of information precision to be effective. This is an important property of loose-coupled networks that can greatly reduce the complexity/overhead of the service connections.

When considering linking capabilities together within an enterprise the default path is usually some form of shared technical infrastructure. This may include communication networks (e.g. LAN/WAN) and may also leverage shared database/storage capabilities where attempts are made to have all data conform to a common schema and definition.

But in highly distributed networks where the services can be supported by many different enterprises operating their own business applications and technical platforms, reaching common agreements on a data schema and definitions is a significant challenge. Any opportunity to lower this requirement can be highly advantageous.

Determining the required ‘precision’ for a service exchange evaluates the extent to which a service operation can be realized successfully with more approximate information and data definitions. The BIAN service operations are defined in simple semantic/narrative terms. It then has to be determined for each exchange how far the definition of these business information terms need to be defined and agreed as they are driven down to more detailed specifications for implementation.

An example can help clarify this concept. Two capability service centers (or Service Domains) that exchange financial transaction data such as product fulfillment and transaction accounting clearly need to have agreed the meaning of the service content with a very high level of precision (amounts, dates, currencies etc. will need to share common data schema and definitions). Elsewhere two Service Domains exchanging marketing insights may share intelligence with a much lower level of precision. A marketing Service Domain may identify a number of prospects and wish to notify a sales Service Domain. There are several aspects of this exchange where the need for machine level data precision can be relaxed:

- ***Multiple/redundant references*** – the service may identify the candidates using a number of characteristics/properties that can be used in different combinations to allow the association to be made (e.g. name, DOB, address, social security number, internal customer reference)
- ***Formatting variations*** – the representation of business information elements may be matched in a manner that handles differences (different field lengths/sequencing, missing optional fields, etc.)
- ***Different data schema*** – the internal data storage for two Service Domains can be different as long as they can both trace their internal data representations to the shared semantic terms with the necessary precision
- ***Acceptance of errors*** – it may also be that the service can tolerate a certain level of errors in the exchange – if for example only 99 of 100 prospects are recognized, the service exchange may still be considered acceptable

The example highlights that service exchange situations where reduced precision is adequate is in the areas of the business outside the highly automated transaction processing core. Interestingly these are areas where new business models are encouraging greater collaboration between enterprises, with more banks exploiting specialized third party intelligence provision and analytical services. The different degrees of required precision for service operation exchanges are shown by simple example in the Figure below:

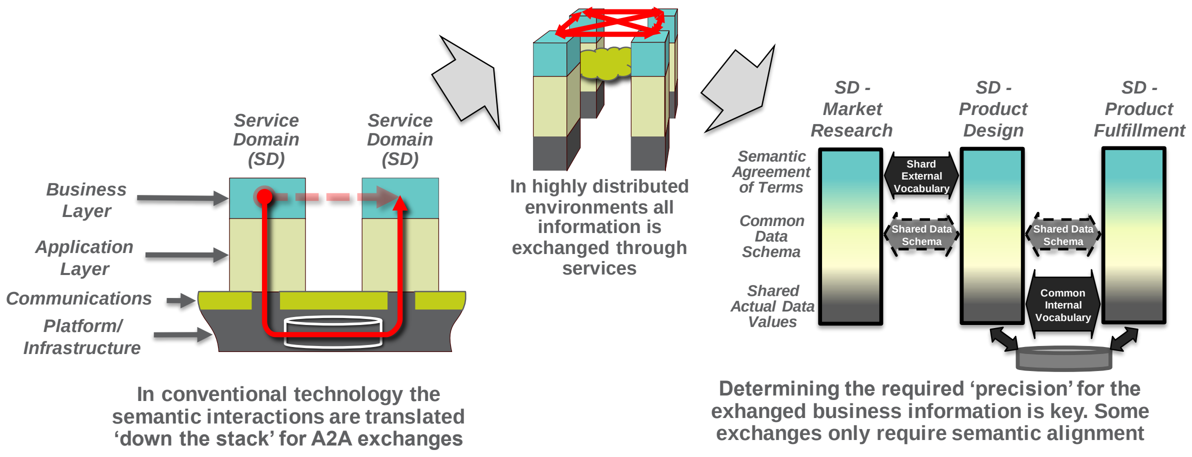

Service Domain (SD)
Service Domain (SD)
Business Layer
Application Layer
Communications
Platform/ Infrastructure
In conventional technology the semantic interactions are translated 'down the stack' for A2A exchanges
In highly distributed environments all information is exchanged through services
SD - Market Research
SD - Product Design
SD - Product Fulfillment
Semantic Agreement of Terms
Common Data Schema
Shared Actual Data Values
Shard External Vocabulary
Shared Data Schema
Shared Data Schema
Common Internal Vocabulary
Determining the required 'precision' for the exhanged business information is key. Some exchanges only require semantic alignment

Figure 15: Matching service operations to the required level of precision

There are already many commercially available cloud based production banking solutions in CRM and risk management. The operational behaviors enabled by the technology are particularly well suited to the networked/collaborative nature of the operational activities in these areas.

The progressive refinement and adoption of the BIAN standard will help encourage the development of standard solution designs that will improve re-deployment and integration activities for cloud and the more recently introduced of micro-service architecture based offerings in the banking industry in general.

Using BIAN Service Domain partitions to define cloud based services

BIAN published a white paper that describes how BIAN Service Domains provide a template for defining and managing cloud based services. This can include providing application program interface (API) based access to these services. The white paper can be found at www.BIAN.org.

A key consideration explained in the white paper addresses how access to the bank’s offered services can be controlled. Many banks are looking for ways to move beyond offering ‘packaged’ products and services to their customers and providing some form of direct access to their core banking capabilities (e.g. payments, cash management, asset & liability management, credit and risk management, financial relationship management). This allows customers and service alliance partners to find innovative ways to integrate the bank directly into their operations.

Bank’s wanting to provide external access to their capabilities need to find a way to do this in a secure manner. Achieving adequate security spans many levels, including the platform and application access controls (IAAS/PAAS). These are complicated and necessary aspects to address and solutions are available or rapidly emerging. The security aspect of external service provision that BIAN can help with is to define the allowed use of offered services in a standard form (SAAS).

When services are consumed within an enterprise it can be assumed and assured that the services are being accesses by suitably authorized parties and are being used for appropriate purposes. When the service is made available to an external party these constraints are far harder to achieve. It is also desirable to find a way to offer external services in a standard way so the services can be made widely available under standard controls without the need to implement each external connection individually.

A way to approach this requirement is to use the business purpose/role of the Service Domain to define the service operation business context and then use this to control/constrain service access. The Figure below shows an example where the Relationship Management Service Domain is used to provide the business context. The services made available reflect the allowed use for a relationship manager (probably further constrained for an external relationship manager).

For example, the ‘relationship manager’ is only able to access ‘their’ customers’ details and only access (and update) information pertinent to the role of relationship manager. The times of allowed access and volumes/frequency may also be limited to that appropriate for a relationship management position.

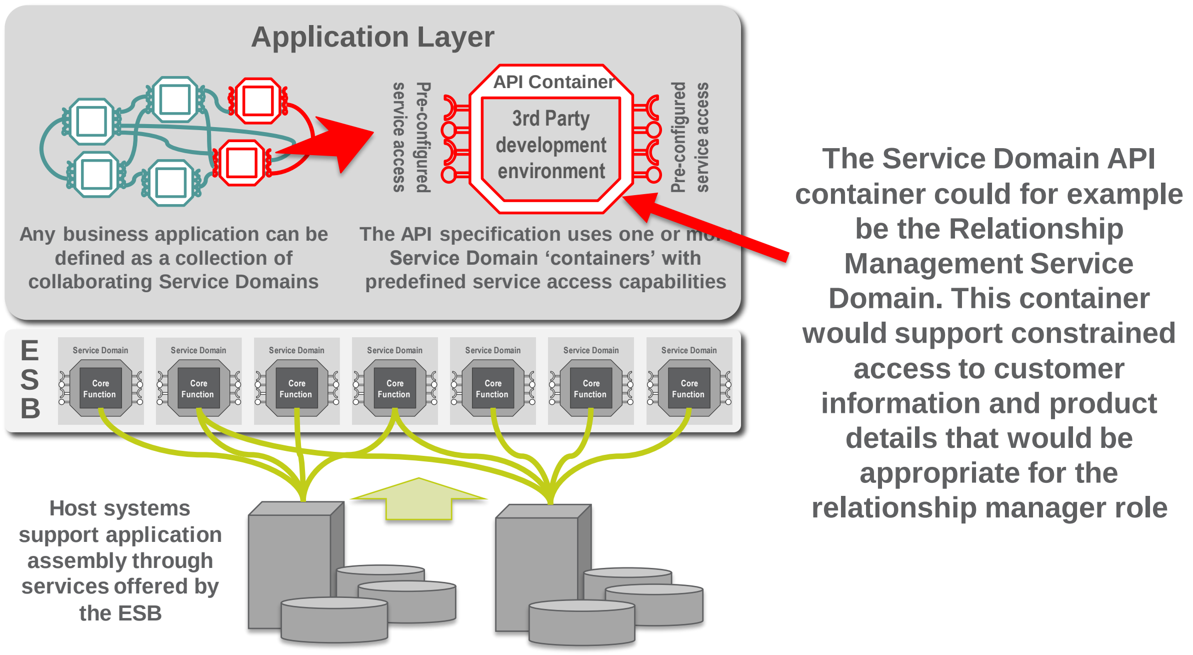

Application Layer
Pre-configured service access
API Container
3rd Party development environment
Pre-configured service access
Any business application can be defined as a collection of collaborating Service Domains
The API specification uses one or more Service Domain ‘containers’ with predefined service access capabilities
The Service Domain API container could for example be the Relationship Management Service Domain. This container would support constrained access to customer information and product details that would be appropriate for the relationship manager role
E S B
Service Domain
Core Function
Host systems support application assembly through services offered by the ESB

Figure 16: Cloud based services for a Relationship Management Service Domain

Application Programming Interfaces (APIs)

This use of the BIAN Service Domain acting as a service ‘container’ that defines a limiting context for access is a way to help define standard application program interface (API) access capabilities. The container and its associated service APIs offers constrained access to the bank and the third party can develop differentiated functional capabilities within the container(s). The use of BIAN to support the development of APIs is addressed later in this section.

Cross Technical Platform Solutions

As noted earlier, most banks have application portfolios that combine technologies corresponding to all of the three types of environment just described. An advantage of the BIAN service landscape it that it can be interpreted consistently in any of these technical environments. The Service Domains define conceptual/logical designs with functional partitions that are the same in each case. Accordingly, the model can facilitate the integration of business applications that may be developed and operate in these different environments. For example, using the BIAN service operation standards it is possible to find ways to sensibly integrate external cloud based services with internal legacy host transaction processing systems.

2.6 Specifying Point Solution Requirements – Accelerator Packs

This sub-section describes the typical steps that can be followed to apply the BIAN designs in the context of a targeted or point solution. A project at BIAN member PNC Bank provides a particularly good case study following this approach that can be found on www.BIAN.org.

The templates combined with the core BIAN definitions provide a very re-useable collection of general designs. The canonical nature of the BIAN Service Domains and service operations ensure that the designs can be consistently interpreted. Solutions targeting a particular area (in the examples included in this section this area is that of the payments function) provide a template that can be re-applied elsewhere.

The working name for the collection of designs covering an area of business operation is an ‘Accelerator Pack’. The BIAN specifications have been extended since the case study referenced in this section and the approach updated accordingly. The stages applying BIAN in the point solution approach described here are as follows:

1. **Business Case** – structured analyses to assess the likely business impact of the solution are used to define the business case for investment
2. **Business Scenarios** – the boundary/scope and main internal activities are captured using Business Scenarios
3. **Wireframe** – the Business Scenarios are used to assemble a ‘wireframe’ model that shows the involved Service Domains and primary service connections in a stable framework
4. **Requirements** – the functional and non-functional requirements for the Service Domains in scope are developed. If necessary, the semantic service operation definitions are extended. These define the implementation level requirements and may include site specific variations
5. **Solution Mapping** – current systems and/or candidate packages and new development options are mapped against the wireframe and more detailed implementation requirements (aligned to the Service Domains)
6. **Customization/Development** – identified business application and interface shortfalls are outlined and customization and development tasks/options scoped out
7. **Deployment Planning** – finally a phased, multithreaded migration project is detailed with the necessary business case for investment consideration

The earlier description of different technical environments can be referenced to adapt this approach as necessary for different technical situations. Each stage is described below:

2.6.1 Business Case Development

Conventional business case development techniques can be used to estimate the financial impact of current shortfalls and anticipate the likely impact on performance arising from the proposed development. This analysis may identify different general categories of business performance (such as productivity, error rates, time to market) and compare current and future sate impact in terms of measurable cost savings, improved net revenues and reduced risk exposure.

The business case may also reference the more specific benefits of adopting service oriented approaches and the particular BIAN SOA properties described earlier in Section 2.1 of this guide.

2.6.2 Select and Amend Business Scenario(s)

Business Scenarios are used to define the main business events that the point solution will handle; such as processing product fulfillment tasks, handling a customer inquiry, processing a payment transaction. An example PowerPoint view of a BIAN Business Scenario is shown:

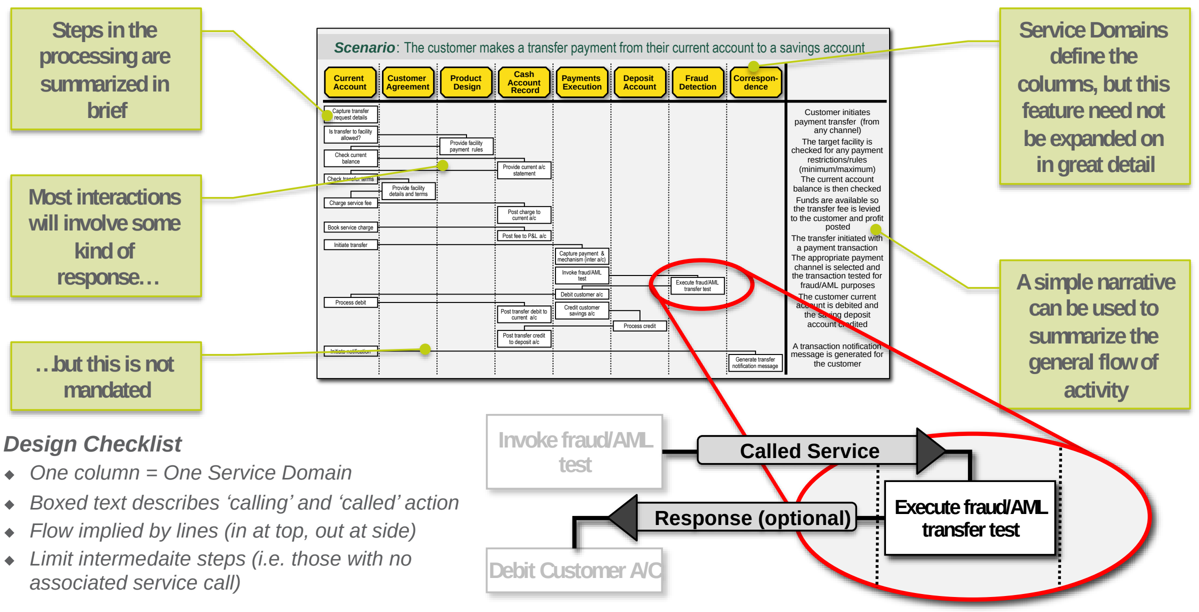

Steps in the processing are summarized in brief
Most interactions will involve some kind of response…
…but this is not mandated
Scenario: The customer makes a transfer payment from their current account to a savings account
Current Account
Customer Agreement
Product Design
Cash Account Record
Payments Execution
Deposit Account
Fraud Detection
Correspon-dence
Capture transfer request details
Is transfer to facility allowed?
Provide facility payment rules
Check current balance
Provide current a/c statement
Check transfer terms
Provide facility details and terms
Charge service fee
Post charge to current a/c
Book service charge
Post fee to P&L a/c
Initiate transfer
Capture payment & mechanism (inter a/c)
Invoke fraud/AML test
Execute fraud/AML transfer test
Debit customer a/c
Process debit
Post transfer debit to current a/c
Credit customer savings a/c
Process credit
Post transfer credit to deposit a/c
Initiate notification
Generate transfer notification message
Customer initiates payment transfer (from any channel)
The target facility is checked for any payment restrictions/rules (minimum/maximum)
The current account balance is then checked
Funds are available so the transfer fee is levied to the customer and profit posted
The transfer initiated with a payment transaction
The appropriate payment channel is selected and the transaction tested for fraud/AML purposes
The customer current account is debited and the saving deposit account credited
A transaction notification message is generated for the customer
Service Domains define the columns, but this feature need not be expanded on in great detail
A simple narrative can be used to summarize the general flow of activity
Design Checklist
One column = One Service Domain
Boxed text describes 'calling' and 'called' action
Flow implied by lines (in at top, out at side)
Limit intermedaite steps (i.e. those with no associated service call)
Invoke fraud/AML test
Called Service
Response (optional)
Execute fraud/AML transfer test
Debit Customer A/C

Figure 17: Example business scenario with rules

Each Business Scenario shows the involved Service Domains and the major service interactions between them. The format and modeling approach for developing Business Scenarios is described in the BIAN 'How-to Guide – Developing Content.'

Example business scenarios can be selected from the BIAN repository and refined and new scenarios developed as needed. The business scenarios should be walked through with business practitioners to confirm they collectively cover all of the key functional requirements of the target area.

2.6.3 Develop a Wireframe model

The Wireframe model is an informal representation of the Service Domain and service connections between them. One or more wireframe views can be developed to capture the activities covered by the point solution. Unlike Business Scenarios that capture a dynamic behavior such as the sequence of tasks needed to process a request, the wireframe is a static framework and many possible dynamic behaviors can be traced or overlain on this framework as shown in the following Figure where a SWIFT payment scenario is traced over a wireframe:

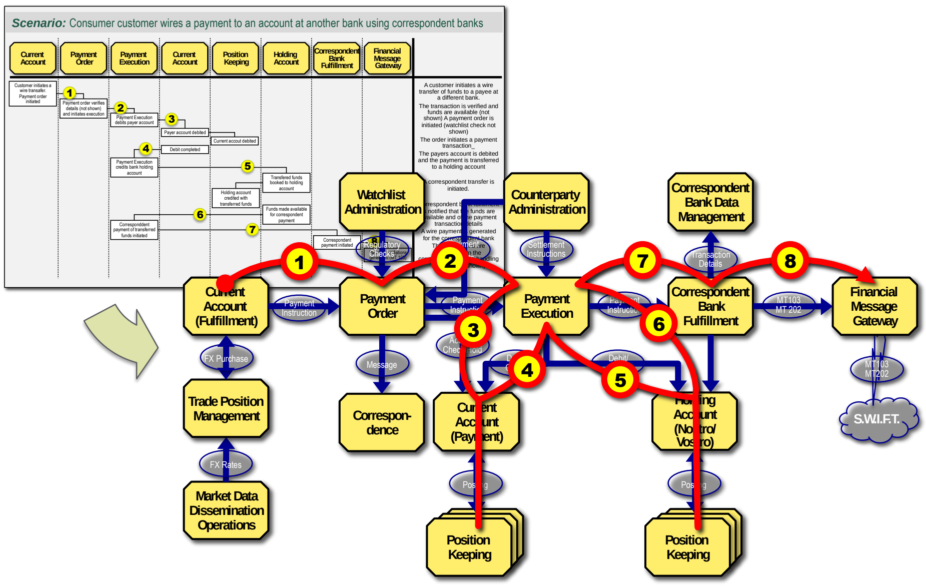

Scenario: Consumer customer wires a payment to an account at another bank using correspondent banks
Watchlist Administration
Counterparty Administration
Correspondent Bank Data Management
Current Account (Fulfillment)
Payment Instruction
Payment Order
Payment Execution
Correspondent Bank Fulfillment
MT103 MT 202
Financial Message Gateway
FX Purchase
Trade Position Management
FX Rates
Market Data Dissemination Operations
Message
Correspondence
Current Account (Payment)
Holding Account (Nostro/ Vostro)
Position Keeping
S.W.I.F.T.
Settlement Instructions
Transaction Details
Regulatory Checks
1
2
3
4
5
6
7
8

Figure 18: A payment transaction mapped on a Wireframe view

The wireframe is particularly useful to agree the scope of the point solution – i.e. the Service Domains (and associated functionality) that is to be supported. It also shows Service Domains for which there is a direct interface. Below the stylized payments wireframe used in the case study is shown. Service Domains have been color coded to show the core Service Domains, interfaced and peripheral Service Domains.

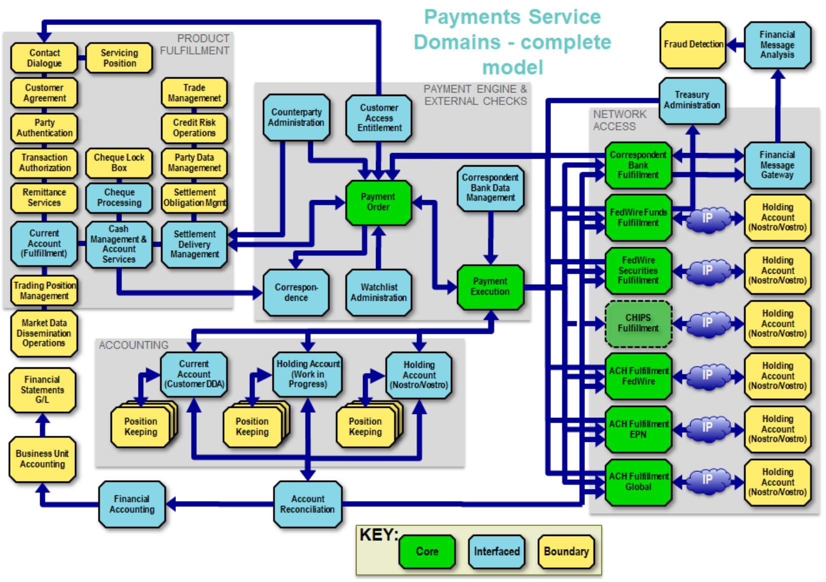

Payments Service Domains - complete model
PRODUCT FULFILLMENT
Contact Dialogue
Servicing Position
Customer Agreement
Trade Management
Party Authentication
Credit Risk Operations
Transaction Authorization
Cheque Lock Box
Party Data Management
Remittance Services
Cheque Processing
Settlement Obligation Mgmt
Current Account (Fulfillment)
Cash Management & Account Services
Settlement Delivery Management
Trading Position Management
Market Data Dissemination Operations
Financial Statements G/L
Business Unit Accounting
PAYMENT ENGINE & EXTERNAL CHECKS
Counterparty Administration
Customer Access Entitlement
Payment Order
Correspondent Bank Data Management
Correspondence
Watchlist Administration
Payment Execution
ACCOUNTING
Current Account (Customer DDA)
Holding Account (Work in Progress)
Holding Account (Nostro/Vostro)
Position Keeping
Financial Accounting
Account Reconciliation
NETWORK ACCESS
Fraud Detection
Financial Message Analysis
Treasury Administration
Correspondent Bank Fulfillment
Financial Message Gateway
FedWire Funds Fulfillment
FedWire Securities Fulfillment
CHIPS Fulfillment
ACH Fulfillment FedWire
ACH Fulfillment EPN
ACH Fulfillment Global
IP
Holding Account (Nostro/Vostro)
KEY:
Core
Interfaced
Boundary

Figure 19: The completed payments area Wireframe (example)

2.6.4 Define the Implementation Requirements

The BIAN Service Domain specifications are high level as appropriate for a business architecture specification. To support implementation activity far more detailed ‘requirement specifications’ are needed. In a prior release the concept of behavior qualifier types was introduced and these have been applied since to break down the working of the Service Domain to a finer level of detail.

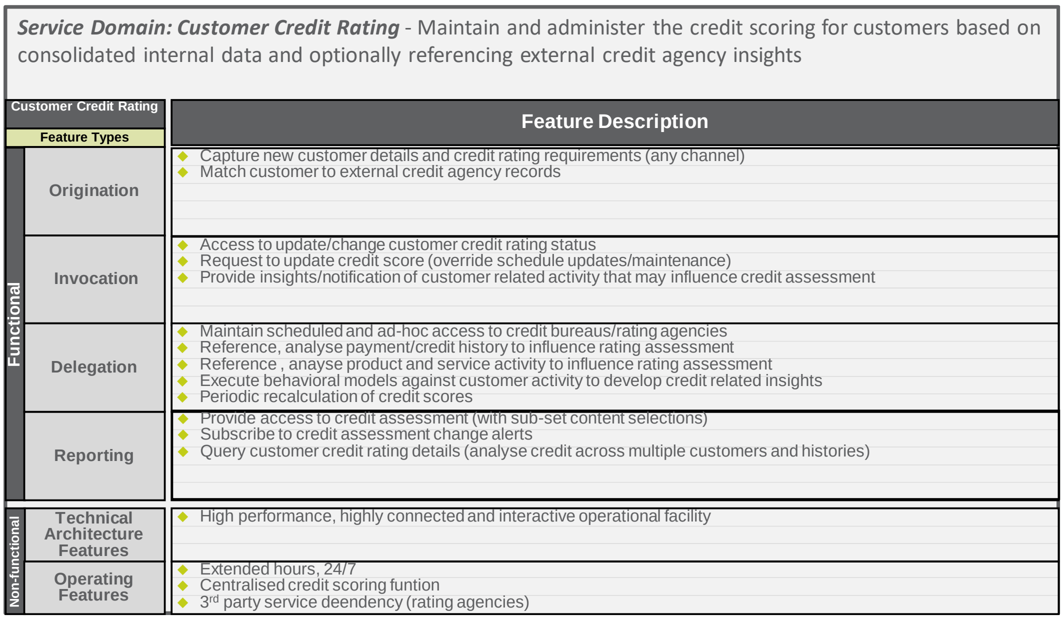
Business application requirements can be captured in many different forms. The BIAN Service Domains, service operations, Business Scenarios and other related templates align well when the target implementation environment is service oriented.

Feature Checklists

The high level description, control record, recently added behavior qualifiers and service operation specifications of the Service Domain can be extended using a Feature Checklist table. The level of detail of the functional features checklists may vary depending on the particular project approach.

For commercial package evaluation the checklist needs to be at a sufficient level of detail to compare/evaluate competing offerings (as might be used in a Request for Proposal or Request for Information (RFP/RFI). If the solution involves new developed or enhancement to existing systems a greater level of detail is likely to be required.

An example Service Domain Feature Checklist is shown for the “Customer Credit Rating” Service Domain. The template organizes the functional features under four ‘responsibility item types’ that are used throughout BIAN to classify aspects of the Service Domain specification for consistency. Key non-functional operational and technical features have been added to the table.



Service Domain: Customer Credit Rating - Maintain and administer the credit scoring for customers based on consolidated internal data and optionally referencing external credit agency insights

Customer Credit Rating		
	Feature Types	Feature Description
Functional	Origination	◆ Capture new customer details and credit rating requirements (any channel) ◆ Match customer to external credit agency records
	Invocation	◆ Access to update/change customer credit rating status ◆ Request to update credit score (override schedule updates/maintenance) ◆ Provide insights/notification of customer related activity that may influence credit assessment
	Delegation	◆ Maintain scheduled and ad-hoc access to credit bureaus/rating agencies ◆ Reference, analyse payment/credit history to influence rating assessment ◆ Reference , anayse product and service activity to influence rating assessment ◆ Execute behavioral models against customer activity to develop credit related insights ◆ Periodic recalculation of credit scores
	Reporting	◆ Provide access to credit assessment (with sub-set content selections) ◆ Subscribe to credit assessment change alerts ◆ Query customer credit rating details (analyse credit across multiple customers and histories)
Non-functional	Technical Architecture Features	◆ High performance, highly connected and interactive operational facility
	Operating Features	◆ Extended hours, 24/7 ◆ Centralised credit scoring funtion ◆ 3rd party service deendency (rating agencies)

Figure 20: Feature list for a Service Domain - Customer Credit Rating

The Feature Checklist table can be adapted to include more detailed specializations that may identify geopolitical, scale and maturity specific features. BIAN does not currently maintain Feature Checklists for the Service Domains. The tables need to be defined on a case by case basis.

Service Operations

In addition to expanding the functional feature lists for the Service Domains, their associated service operation definitions are used to specify the interfaces or services depending on the technical arrangement. The BIAN service operations are intended to include sufficient semantic content to be mapped to the underlying messages. Where industry standard messages exist these mappings may be available from the BIAN repository. The approach for mapping the service operations to messages was described earlier in this guide. Depending on the target technical environment the BIAN service operations can be interpreted as the interface that can be 'hard-wired' or can be implemented as a fully functional service operation.

For the Service Domains that are a part of the target solution all service operations as defined for the Service Domain will need to be supported to handle all possible (life-cycle) behaviors. For the services that are accessed by the target solutions the service operations referenced in the range of Business Scenarios define the type of external connections/interfaces needed for the point solution.

As noted earlier the BIAN service operations simply capture the business information exchange dependencies in semantic terms. They do not define in any detail the nature or choreography of the underlying message exchange (i.e. simple one/two-way exchange or lengthy negotiation/dialogue). The requirements specification will clearly need to specify this aspect of the service implementation to help with sizing and designing the external interfaces.

Business Scenarios & Wireframes

The collection of Business Scenarios and any derived Wireframe views define requirements by example. The range of scenarios typically provides the detail for the main business events. It will not be comprehensive, for example it will not usually cover error and exception handling.

Where more detail is required additional Business Scenarios can be developed or more conventional requirement definition approaches used to augment the service based requirements produced using the BIAN Business Scenarios, Service Domain feature lists and service operation definitions.

Taken together the Feature Checklists, Service Operations, Business Scenarios and other information templates provide a framework for defining the high level requirements for a point solution. This framework can be expanded adding detail to the Service Domain partitions if the solution is service oriented or any other suitable format can be adopted to support the particular implementation environment as necessary.

2.6.5 Map and Assess Existing Systems/Candidate Packages

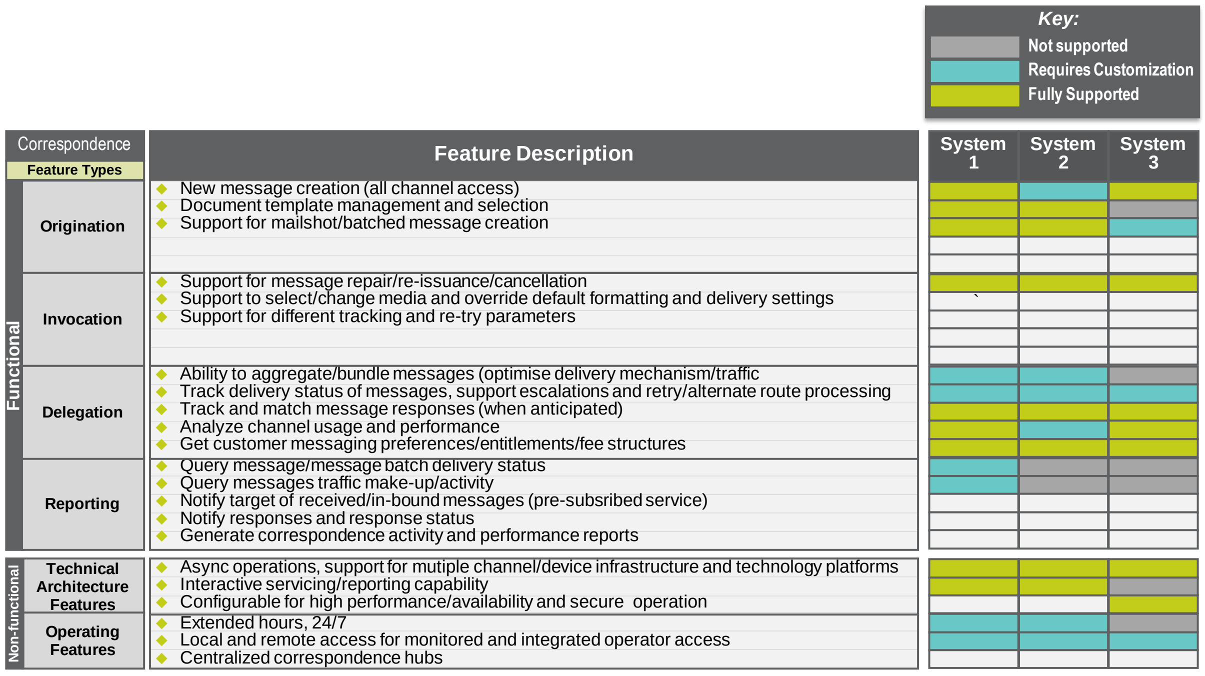
Existing systems and external candidate packages are mapped in the same way. The assessment of existing business applications and candidate commercial offerings combines several factors

1. ***Functional Coverage*** – how well are the functional needs supported?
2. ***Service Enablement*** – how well does the candidate solution align to and support service based operations?
3. ***Hygiene Factors*** – are there any 'deal breakers' that severely constrain the use of a candidate system?

Functional Coverage

The mechanism to check coverage is the Service Domain feature checklists described earlier in this guide. Because the Service Domains define non-overlapping business capabilities, when solution options are mapped to the features of the Service Domains and compared one Service Domain at a time they provide a mechanism for ensuring comparisons are made considering "like-for-like" elements.

In the Figure below three candidate systems (that could be existing applications or commercial candidates). A simple assessment is performed to determine whether the requirement is either fully supported, can be supported with limited enhancement work or if the requirement is not supported at all.



Key:

▨	Not supported
🟦	Requires Customization
🟩	Fully Supported

Correspondence	Feature Types	Feature Description	System 1	System 2	System 3
Functional	Origination	New message creation (all channel access)	🟩	🟦	🟩
		Document template management and selection	🟩	🟩	▨
		Support for mailshot/batched message creation	🟩	🟩	🟦
	Invocation	Support for message repair/re-issuance/cancellation	🟩	🟩	🟩
		Support to select/change media and override default formatting and delivery settings	`		
		Support for different tracking and re-try parameters			
	Delegation	Ability to aggregate/bundle messages (optimise delivery mechanism/traffic	🟦	🟦	▨
		Track delivery status of messages, support escalations and retry/alternate route processing	🟦	🟦	🟦
		Track and match message responses (when anticipated)	🟩	🟩	🟩
		Analyze channel usage and performance	🟩	🟦	🟩
		Get customer messaging preferences/entitlements/fee structures	🟩	🟩	🟩
	Reporting	Query message/message batch delivery status	🟦	▨	▨
		Query messages traffic make-up/activity	🟦	▨	▨
		Notify target of received/in-bound messages (pre-subsribed service)			
		Notify responses and response status			
		Generate correspondence activity and performance reports			
Non-functional	Technical Architecture Features	Async operations, support for mutiple channel/device infrastructure and technology platforms	🟩	🟩	🟩
		Interactive servicing/reporting capability	🟩	🟩	▨
		Configurable for high performance/availability and secure operation			🟩
	Operating Features	Extended hours, 24/7	🟦	🟦	▨
		Local and remote access for monitored and integrated operator access	🟦	🟦	🟦
		Centralized correspondence hubs			

Figure 21: Mapping candidate systems to the feature list of a Service Domain

Based on this mapping the solution with the best coverage and least required enhancement investment can be selected. Even with the objective assessment of coverage by Service Domain this selection can be a complex process for many reasons. Candidate solutions will typically different combinations of Service Domains and may be assessed to have different relative strengths. Furthermore, it may be difficult to split away parts of monolithic systems.

The checklist Service Domain mapping is useful for highlighting specific functional shortfalls. Mapping using the Wireframe view can be used to expose major overlaps/conflicts and the external boundary interfacing needs.

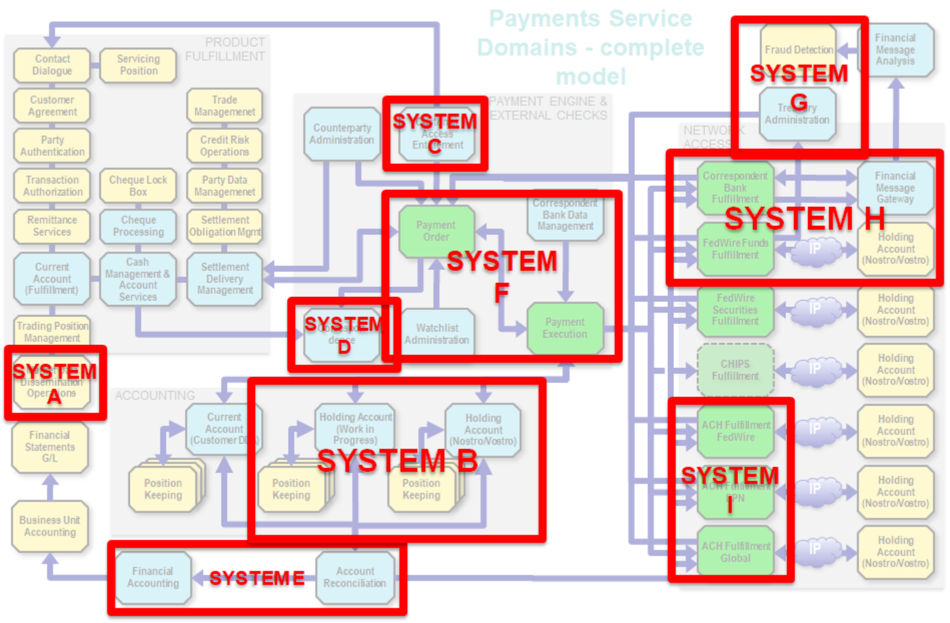


Figure 22: Overlaying current systems on a Wireframe model

Service Enablement

An additional test for the mapped candidate solutions against the Service Domains considers their ‘service alignment’. There are two main considerations:

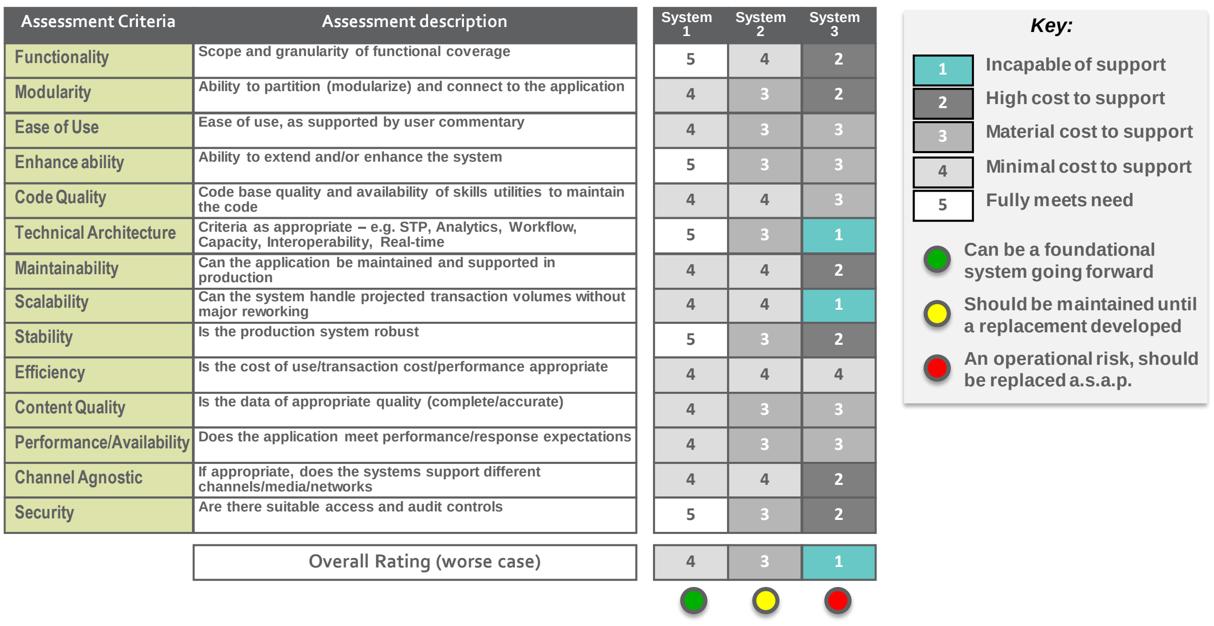
> ***Supporting service operations*** – do they (the candidate systems) support the required collection of offered service operations? Existing interfaces may need to be improved to generalize and service enable them so that they can be called by any suitable party.
>
> ***Externalizing duplicated functionality*** – considering the delegated services for the Service Domain, does the mapped system make external calls to fulfill these needs or is the functionality embedded and what would be necessary to externalize this capability when necessary?

For larger monolithic systems it may not always be easy to expose the full range of operational services for all ’contained’ Service Domains.

Candidate System 'Hygiene Factor Analysis'

Hygiene Factor Analysis can be useful to quickly eliminate candidate solutions and simplify the selection process. There are many possible factors involved in determining the suitability of a candidate solution (functional, technical and operational) that influence its long term use or strategic alignment. For example, a system that might have good functional coverage today may have severely limited ability for enhancement or to scale up to support higher volumes.

Using this analysis operationally flawed systems that need to be retired are exposed. The system with the best 'strategic fit' is selected. All other systems can be placed on a 'maintenance only' investment footing. An example analysis is shown in the Figure below:



Assessment Criteria	Assessment description	System 1	System 2	System 3
Functionality	Scope and granularity of functional coverage	5	4	2
Modularity	Ability to partition (modularize) and connect to the application	4	3	2
Ease of Use	Ease of use, as supported by user commentary	4	3	3
Enhance ability	Ability to extend and/or enhance the system	5	3	3
Code Quality	Code base quality and availability of skills utilities to maintain the code	4	4	3
Technical Architecture	Criteria as appropriate – e.g. STP, Analytics, Workflow, Capacity, Interoperability, Real-time	5	3	1
Maintainability	Can the application be maintained and supported in production	4	4	2
Scalability	Can the system handle projected transaction volumes without major reworking	4	4	1
Stability	Is the production system robust	5	3	2
Efficiency	Is the cost of use/transaction cost/performance appropriate	4	4	4
Content Quality	Is the data of appropriate quality (complete/accurate)	4	3	3
Performance/Availability	Does the application meet performance/response expectations	4	3	3
Channel Agnostic	If appropriate, does the systems support different channels/media/networks	4	4	2
Security	Are there suitable access and audit controls	5	3	2
	Overall Rating (worse case)	4	3	1
		🟢	🟡	🔴

Key:

1	Incapable of support
2	High cost to support
3	Material cost to support
4	Minimal cost to support
5	Fully meets need
🟢	Can be a foundational system going forward
🟡	Should be maintained until a replacement developed
🔴	An operational risk, should be replaced a.s.a.p.

Figure 23: Example hygiene factor analysis

Some points of note for interpreting the analysis shown in the above figure:

- The sliding scale can be used to identify the best fit system for a Service Domain
- A system that scores an exception score on any factor represents an operational risk regardless of its scores on all other criteria and should be retired.
- A system that is not selected as the strategic fit should only be maintained (i.e. no investment to extend the system), as it will be replaced. All technical alignment and functional enhancement investment should be directed to the strategic/best fit system.

2.6.6 Customization/Development

The Service Domain designs, in particular the Feature Checklists have been described in terms of solution selection. The same mechanism can be adapted and used to specify enhancement and customization needs and used to track the implementation of these developments.

2.6.7 Migration Planning

The Service Domain partitions provide an opportunity to structure the migration plan by providing functional streams that can be addressed in parallel. One or more associated Service Domains can define one development thread. The service dependencies between Service Domains can also be used to help identify any critical dependencies (pre/post dependencies) between the parallel development threads.

In addition, the Service Domain aligned requirements can be used to identify phases in the migration, where coordinated development across the parallel development threads can provide some meaningful interim phased deliverable. The helps releasing benefits in incremental phases during the migration that can offset the overall program investment.

Multi-threaded, multi-phase migration planning is a technique that is widely used. The Service Domain partitions define useful elements in the definition of such a plan.

2.7 Semantic API Initiative

This Section repeats extracts from descriptions of the BIAN API Initiative that can be found in the How To Guide – Developing Content and also in more complete form in the BIAN Semantic API How to Guide. BIAN is also considering the development of a more detailed practitioners guide to augment on-line guidance that is integrated into the BIAN API Exchange.

The Semantic API Initiative is a major ongoing undertaking within BIAN to develop and package BIAN specifications in a form that can be used for ‘aligned’ API development. With the latest release BIAN has published extended Service Domain and service operation specifications and business object model views for areas of the landscape covered by the first cycle or ‘Wave 1’ of this initiative. Wave 1 content covers:

- Mobile access – direct customer access and via a third party agent. This includes various security and routing capabilities
- Customer On-boarding – the procedures to establish a new customer and the offer process
- Payments – specific focus on consumer payment activity as covered by the European PSD2 initiative
- Consumer Loan – the basic access to a simple unsecured consumer loan

Subsequent waves are planned to extend coverage across all main product and service activities of the landscape.

The API initiative combines three main deliverables:

1. A BIAN API Directory – that uses the BIAN Service Landscape, Service Domain and service operations to classify available APIs for reference
2. Extended BIAN Specifications – to support API development
3. The extended specification is translated into RESTful API endpoint specifications available through the BIAN API Exchange

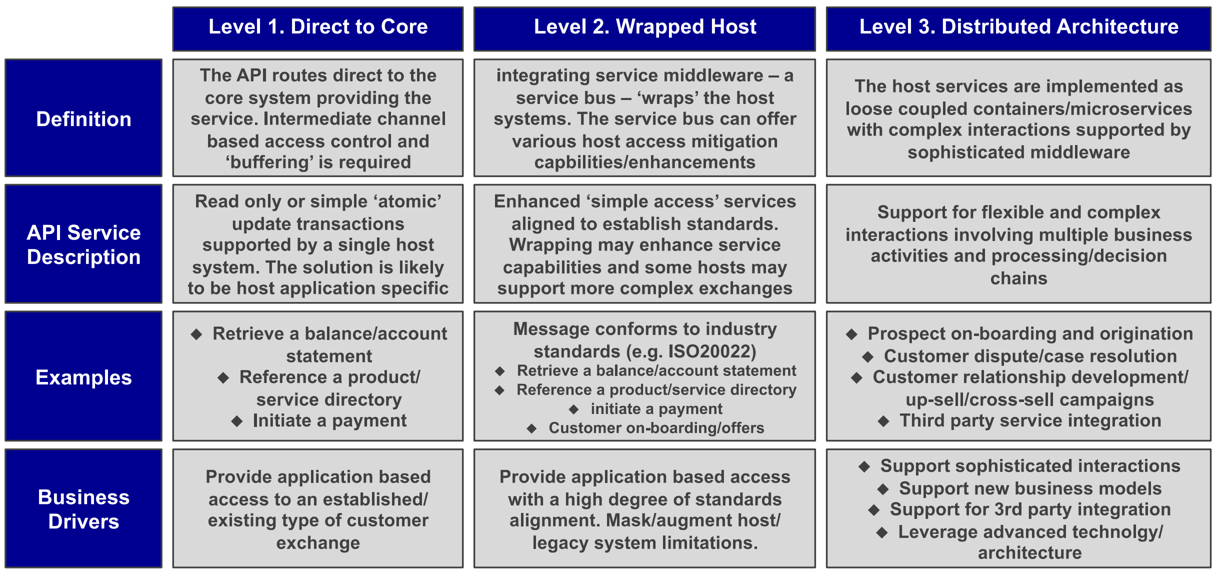
These are supported by a How to Guide and as noted additional practitioner guidance that may be delivered through on-line content in the exchange and/or a published practitioner guide of some form. The deliverables are outlined briefly in this guide but first an explanation is given for three types of BIAN aligned API implementation.

2.7.1. Three Types of Approach in the deployment of APIs

The implementation of open APIs varies greatly depending on the technical architectural approach adopted. BIAN defines three distinct technical solutions corresponding to differing levels of sophistication.

1. ***Direct to Core*** – the service exchange accesses the host facility directly.
2. ***Wrapped Host*** – the host systems are accessed through a control/wrapping middleware such as an enterprise service bus (ESB)
3. ***Distributed Architecture*** – the applications are implemented as a network of service enabled, discrete capabilities that can support external access directly

Key properties of each level and the business rational for adopting each is summarized in the table:



	Level 1. Direct to Core	Level 2. Wrapped Host	Level 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 capbilities/enhancements	The host services are implemented as loose coupled containers/microservices with complex interactions supported by sophisticated 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 establish 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	◆ Retrieve a balance/account statement ◆ Reference a product/ service directory ◆ Initiate a payment	Message conforms to industry standards (e.g. ISO20022) ◆ Retrieve a balance/account statement ◆ Reference a product/service directory ◆ initiate a payment ◆ Customer on-boarding/offers	◆ 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.	◆ Support sophisticated interactions ◆ Support new business models ◆ Support for 3rd party integration ◆ Leverage advanced technolgy/ architecture

Figure 24: Summary table of the BIAN API types of approach

2.7.1.1. Direct to Core

The first type and probably the easiest to implement involves constructing a front-end capability to manage external access security and then typically re-package 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:

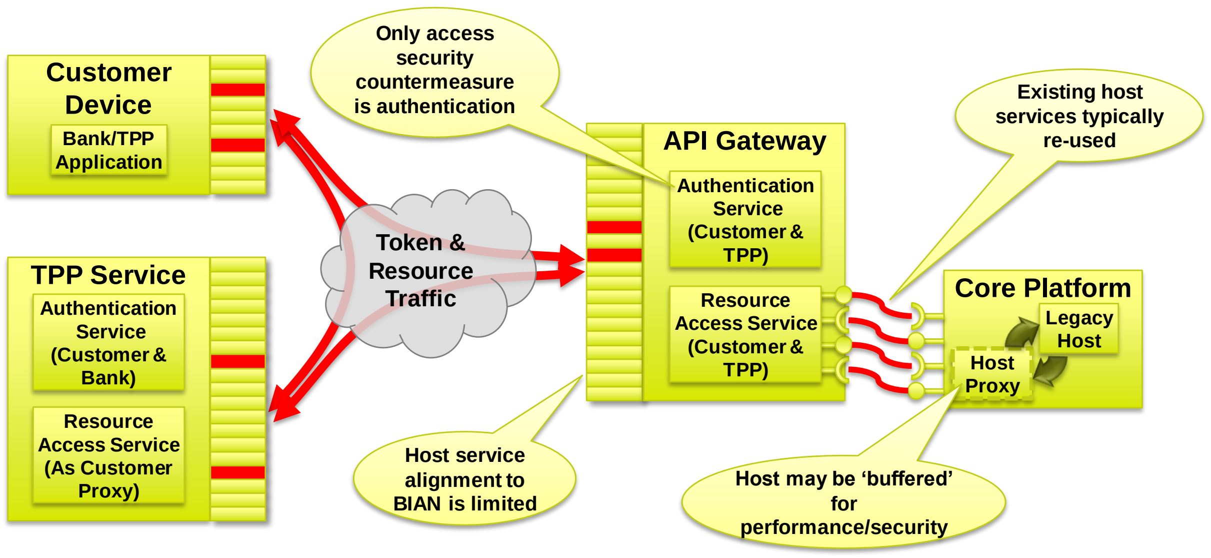

Customer Device
Bank/TPP Application
TPP Service
Authentication Service (Customer & Bank)
Resource Access Service (As Customer Proxy)
Only access security countermeasure is authentication
Token & Resource Traffic
API Gateway
Authentication Service (Customer & TPP)
Resource Access Service (Customer & TPP)
Existing host services typically re-used
Core Platform
Legacy Host
Host Proxy
Host service alignment to BIAN is limited
Host may be 'buffered' for performance/security

Figure 25: Type 1 - layout

Key Aspects of the Approach

At this level 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. 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 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.

2.7.1.2. Wrapped Host

The second type involves the integration of a host access middleware 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

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

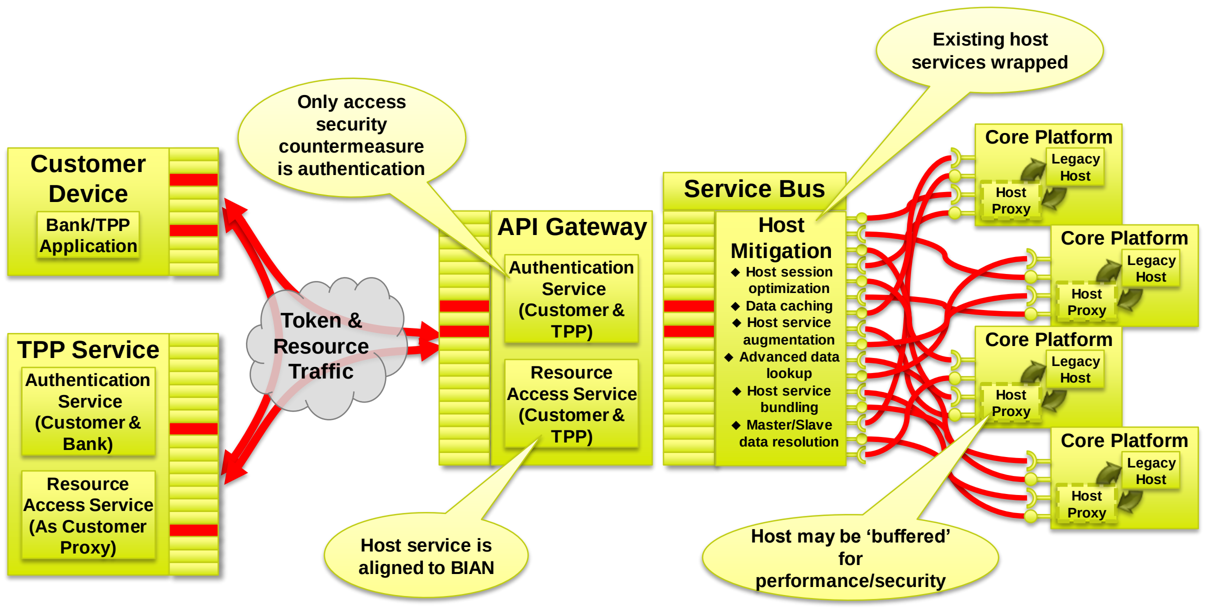

Existing host services wrapped
Only access security countermeasure is authentication
Customer Device
Bank/TPP Application
TPP Service
Authentication Service (Customer & Bank)
Resource Access Service (As Customer Proxy)
Token & Resource Traffic
API Gateway
Authentication Service (Customer & TPP)
Resource Access Service (Customer & TPP)
Service Bus
Host Mitigation
◆ Host session optimization
◆ Data caching
◆ Host service augmentation
◆ Advanced data lookup
◆ Host service bundling
◆ Master/Slave data resolution
Core Platform
Legacy Host
Host Proxy
Host service is aligned to BIAN
Host may be 'buffered' for performance/security

Figure 26: 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 Figure or considered here in any detail.

2.7.1.3. Distributed Architecture

The third type and most sophisticated approach is where the host systems conform to a micro-services/container based architecture 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 manage customer access, providing comprehensive services including access security, activity tracking and intelligent routing decisions.

A micro-service platform that manages external access can link to different host configurations. The Figure below shows how a customer access micro-service platform allows managed access to host systems conforming to different types of API approach (Direct to Core, Wrapped Host and Distributed configurations).

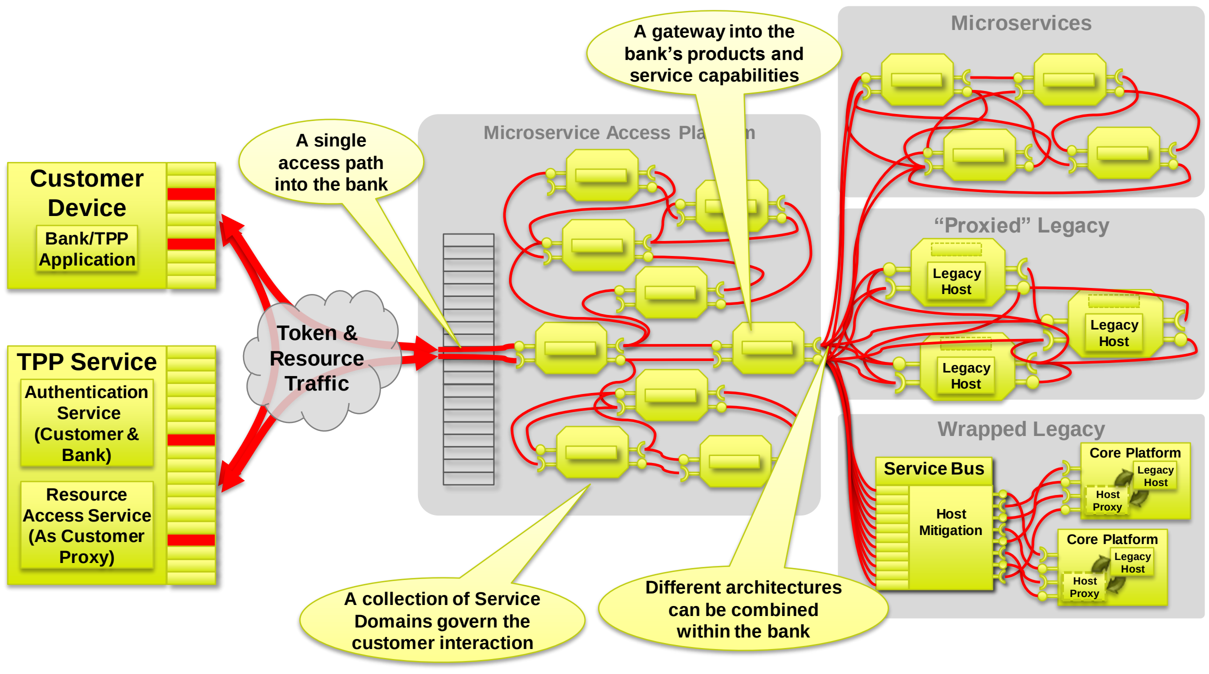

Customer Device
Bank/TPP Application
TPP Service
Authentication Service (Customer & Bank)
Resource Access Service (As Customer Proxy)
A single access path into the bank
Token & Resource Traffic
Microservice Access Platform
A gateway into the bank’s products and service capabilities
Microservices
“Proxied” Legacy
Legacy Host
Legacy Host
Legacy Host
Wrapped Legacy
Service Bus
Host Mitigation
Core Platform
Legacy Host
Host Proxy
Core Platform
Legacy Host
Host Proxy
A collection of Service Domains govern the customer interaction
Different architectures can be combined within the bank

Figure 27: 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 a 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 micro-service/container based architecture where this is appropriate.

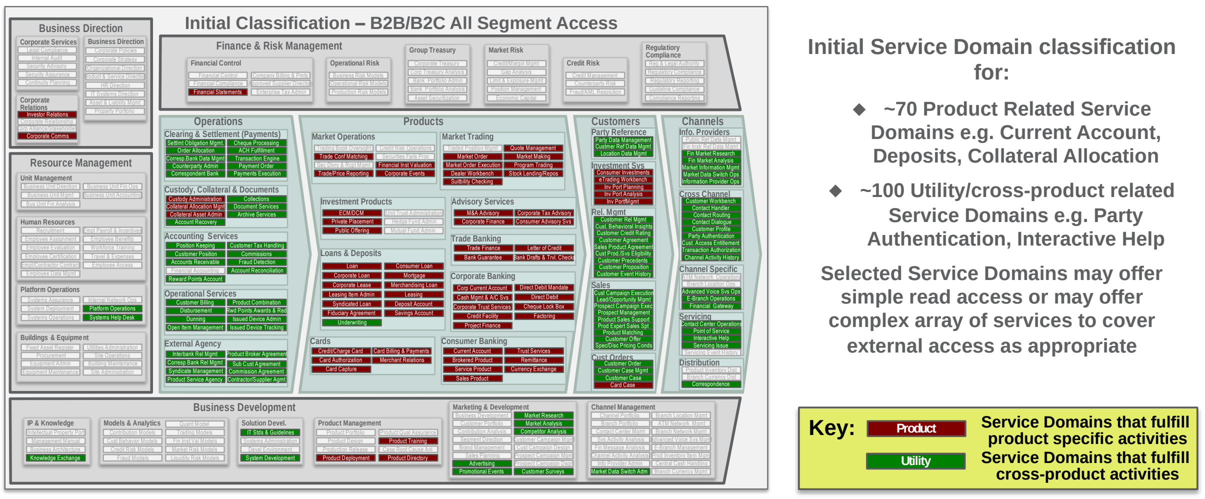
A key advantage of aligning to the BIAN Service Domain and service operation standard for type 1 & 2 solutions is that these interfaces can be later integrated with a type 3 front-end customer access platform with manageable amounts of re-working.

2.7.2. The BIAN API Directory

The BIAN Service Landscape represents a complete inventory of Service Domains and their associated service operations. It can be used as an organizing framework for categorizing/classifying available APIs. By mapping an API to the corresponding service operation(s) for the providing Service Domain it is uniquely classified. It is also used to reference the RESTful API specifications developed by BIAN and presented in the BIAN API Exchange

As the inventory is populated with references to available open APIs users will be able to identify potential solutions. 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 Service Landscape below has been color coded to show those Service domains most likely to provide external access (through an API). Service Domains highlight 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.



Initial Service Domain classification for:

- ~70 Product Related Service Domains e.g. Current Account, Deposits, Collateral Allocation
- ~100 Utility/cross-product related Service Domains e.g. Party Authentication, Interactive Help

Selected Service Domains may offer simple read access or may offer complex array of services to cover external access as appropriate

Key:	
Product	Service Domains that fulfill product specific activities
Utility	Service Domains that fulfill cross-product activities

Figure 28: The Service Landscape with Open API candidates

Every Service Domain has an associated set of default service operations. Some of these support command and control activities that are unlikely to be exposed through APIs and so they have not been included. The other service operations are listed with a brief description to help with mapping. These service operation descriptions are being refined as BIAN develops extended definitions of the Service Domains with this API Initiative. The wave 1 content listed earlier has these extended definitions that can be reviewed

As BIAN expands the coverage of its BOM and the extended specifications are used in deployment the service operation descriptions will be updated with more specific information content.

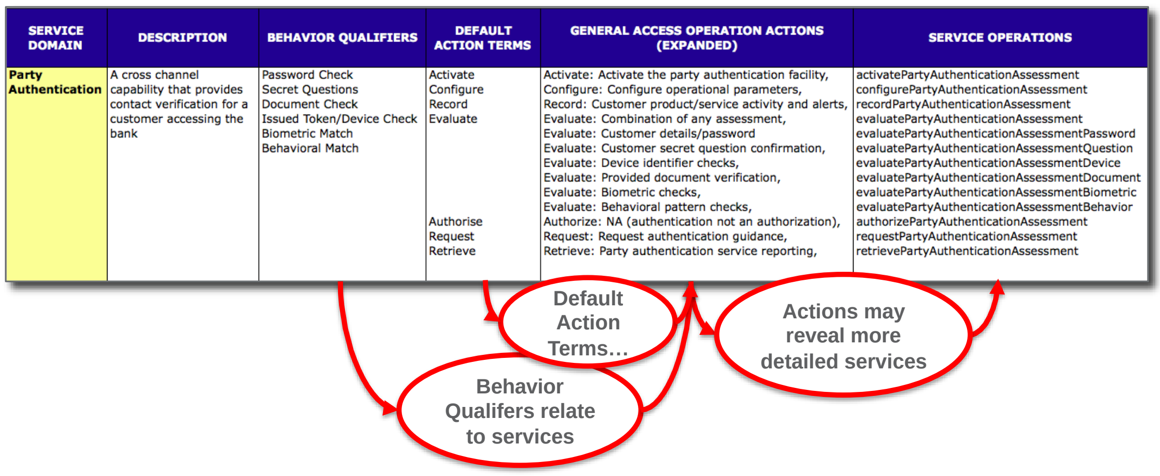
2.7.3 Developing Semantic API Designs

BIAN is developing extended Service Domain and service operation specifications across the Service Landscape. The various design artifacts are outlined below:

- *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. 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 (Touch 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 Exchange

Extended Service Domain Specifications

The additional design concept employed is the 'behavior qualifier type' 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 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 evaluatePartyAuthenticationAssessment evaluatePartyAuthenticationAssessmentPassword evaluatePartyAuthenticationAssessmentQuestion evaluatePartyAuthenticationAssessmentDevice evaluatePartyAuthenticationAssessmentDocument evaluatePartyAuthenticationAssessmentBiometric evaluatePartyAuthenticationAssessmentBehavior authorizePartyAuthenticationAssessment requestPartyAuthenticationAssessment retrievePartyAuthenticationAssessment

Default Action Terms…
Actions may reveal more detailed services
Behavior Qualifers relate to services

Figure 29: Extended Service Domain Specifications (Excel)

Wireframes (showing enterprise boundaries)

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.

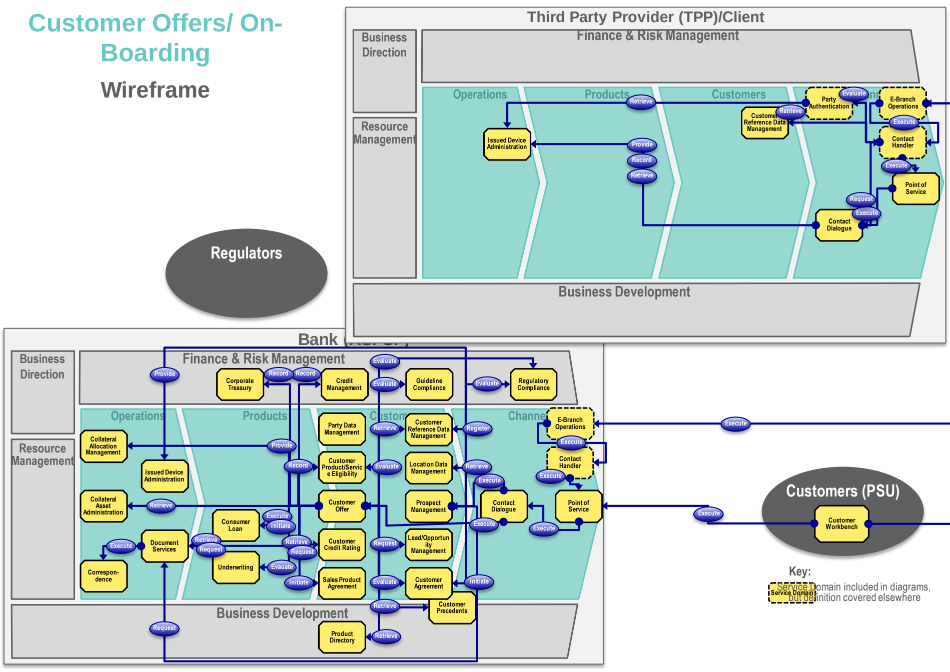

Customer Offers/ On-Boarding
Wireframe
Third Party Provider (TPP)/Client
Finance & Risk Management
Business Direction
Resource Management
Operations
Products
Customers
Business Development
Regulators
Bank
Customers (PSU)
Customer Workbench
Key:
Service Domain included in diagrams, but definition covered elsewhere

Figure 30: Wave 1 Wireframe example

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. An example classification of these dependencies is shown in the mobile access wireframe below:

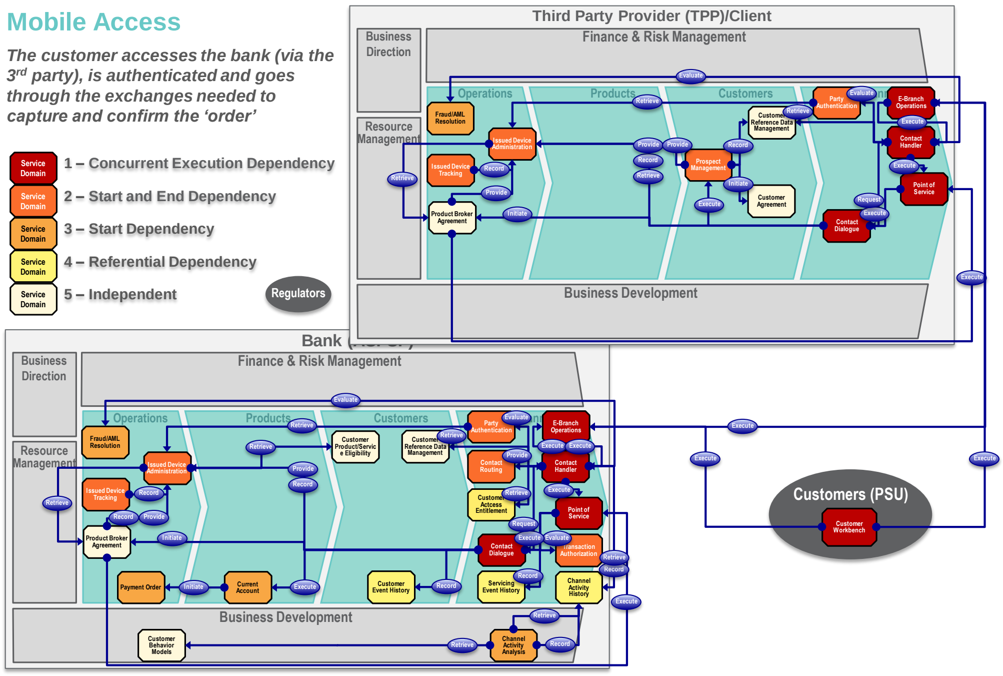

Mobile Access
The customer accesses the bank (via the 3rd party), is authenticated and goes through the exchanges needed to capture and confirm the 'order'
1 – Concurrent Execution Dependency
2 – Start and End Dependency
3 – Start Dependency
4 – Referential Dependency
5 – Independent
Regulators
Third Party Provider (TPP)/Client
Finance & Risk Management
Business Direction
Resource Management
Operations
Products
Customers
Business Development
Bank
Customers (PSU)
Customer Workbench

Figure 31: Mobile Access Wireframe with time dependencies

Enhanced Business Scenarios

For the Semantic API specification, the normal BIAN Business Scenario 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 Figure. 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.

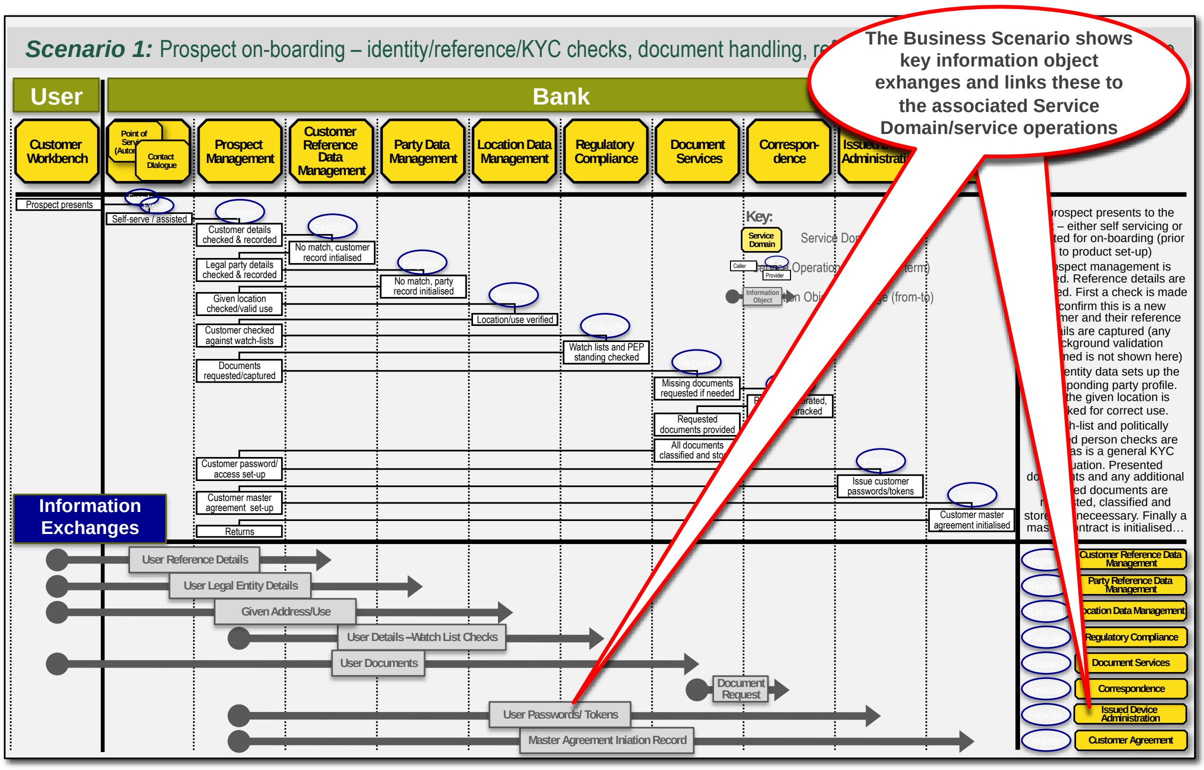

Scenario 1: Prospect on-boarding – identity/reference/KYC checks, document handling,
User
Bank
Customer Workbench
Point of Service
Contact Dialogue
Prospect Management
Customer Reference Data Management
Party Data Management
Location Data Management
Regulatory Compliance
Document Services
Correspon-dence
Issued Device Administration
The Business Scenario shows key information object exhanges and links these to the associated Service Domain/service operations
Prospect presents
Self-serve / assisted
Customer details checked & recorded
No match, customer record initialised
Legal party details checked & recorded
No match, party record initialised
Given location checked/valid use
Location/use verified
Customer checked against watch-lists
Watch lists and PEP standing checked
Documents requested/captured
Missing documents requested if needed
Requested documents provided
All documents classified and stored
Customer password/ access set-up
Issue customer passwords/tokens
Customer master agreement set-up
Customer master agreement initialised
Returns
Key:
Service Domain
Information Object
Information Exchanges
User Reference Details
User Legal Entity Details
Given Address/Use
User Details –Watch List Checks
User Documents
Document Request
User Passwords/ Tokens
Master Agreement Iniation Record
Customer Reference Data Management
Party Reference Data Management
Location Data Management
Regulatory Compliance
Document Services
Correspondence
Issued Device Administration
Customer Agreement

Figure 32: Extended Business Scenario

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. These specifications have been used to define the RESTful APIs presented in the BIAN API Exchange sandbox. With the latest release extended definitions and the associated RESTful API definitions have been defined for 67 Service Domains. BIAN intend to extend this coverage as fast as is practical.

In parallel with creating the extended definitions BIAN is mapping the information content to the BIAN BOM. The BIAN BOM is an extended version of the ISO20022 Business Model, with extensions to map to the BIAN Service Domain and service operation structures and fill gaps in the model as needed. Mapping and extensions have been developed and registered for 16 of the 67 Service Domains at the time of publication.

The initial specification of a service operation is shown in the Excel table:

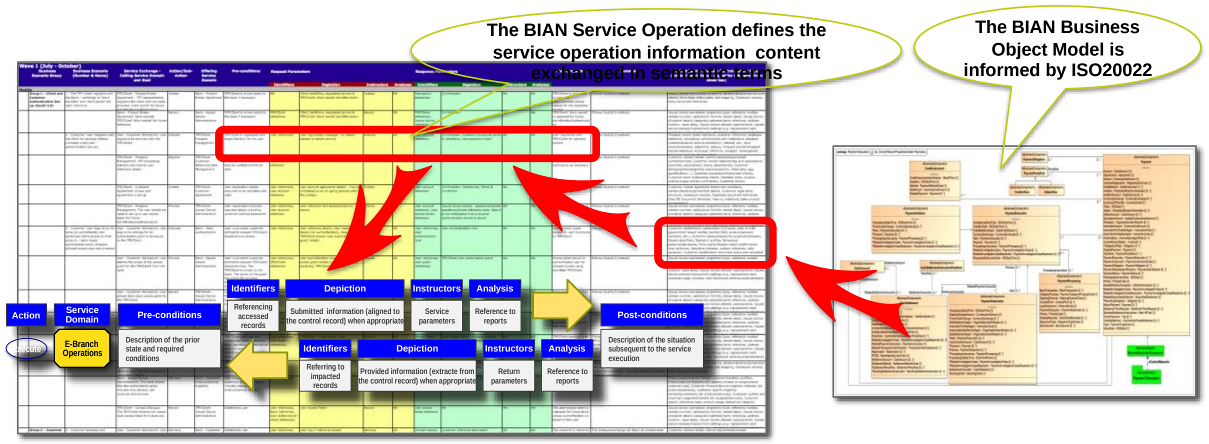

The BIAN Service Operation defines the service operation information content exchanged in semantic terms
The BIAN Business Object Model is informed by ISO20022
Action
Service Domain
Pre-conditions
Execute
E-Branch Operations
Description of the prior state and required conditions
Identifiers
Depiction
Instructors
Analysis
Referencing accessed records
Submitted information (aligned to the control record) when appropriate
Service parameters
Reference to reports
Identifiers
Depiction
Instructors
Analysis
Referring to impacted records
Provided information (extracte from the control record) when appropriate
Return parameters
Reference to reports
Post-conditions
Description of the situation subsequent to the service execution

Figure 33: Service Operation definition (Excel)

2.7.4 Applying the BIAN Semantic API Designs

The Semantic API Working Group is developing practitioner guidance that is integrated into BIAN API Exchange and may also be published in a practitioner guide if necessary. The BIAN Semantic API guide already mentioned is targeted at technical architects who may need to understand the design concepts behind the specification in more detail.

Key aspects of the approach for the different types of API approach already described are now outlined.

For Type 1 – Direct to Core

The approach for applying the BIAN standard for direct to core 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 maps the business service exchange handled by the API as closely as possible to a service connection between two Service Domains. In practice a ‘complete’ API may group together more than one business exchanges mapping to multiple BIAN service connections but the approach is simply repeated for each. The tasks of aligning to the BIAN standard includes the following task:

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. For an expanding proportion of the BIAN landscape the service operations have been translated into the RESTful API specifications in the BIAN API Exchange. Furthermore, in some cases these high level semantic properties are mapped to more detailed definitions provided by the ISO20022 Business Model. The designer needs to select from this list at whatever level of detail is available to populate the underlying request and response messages with the necessary business information content
6. *Standard API Implementation* – from this point on the implementation will be determined by physical and practical requirements specific to the implementation, employing established API design and implementation techniques.

As BIAN members and others develop aligned APIs the confirmed and potentially more detailed service operation specifications can be captured and cross referenced through the BIAN API Exchange. BIAN will also continue to expand the coverage of its own RESTful end point specifications for BIAN Service Domains in the Exchange as fast as is practical.

For Level 2 – Wrapped Host

The approach for a type 2 wrapped host solution builds on the tasks defined for a type 1 solution. 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. 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 – 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 would be 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

For type 3 – Distributed Architecture

The Distributed architecture approach has two distinct aspects. One is the customer access platform that can be implemented using a container/micro-service based solution. The other includes any core banking activities that may be supported using a micro-service architecture.

1 - Micro-service/Container based customer access platform

It is not intended to provide a detailed design of the customer access platform in this guide. The key features as defined using the BIAN Service Domains to represent micro-services are outlined as a starting point.

The customer access platform presents a single point of contact for the user and 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:

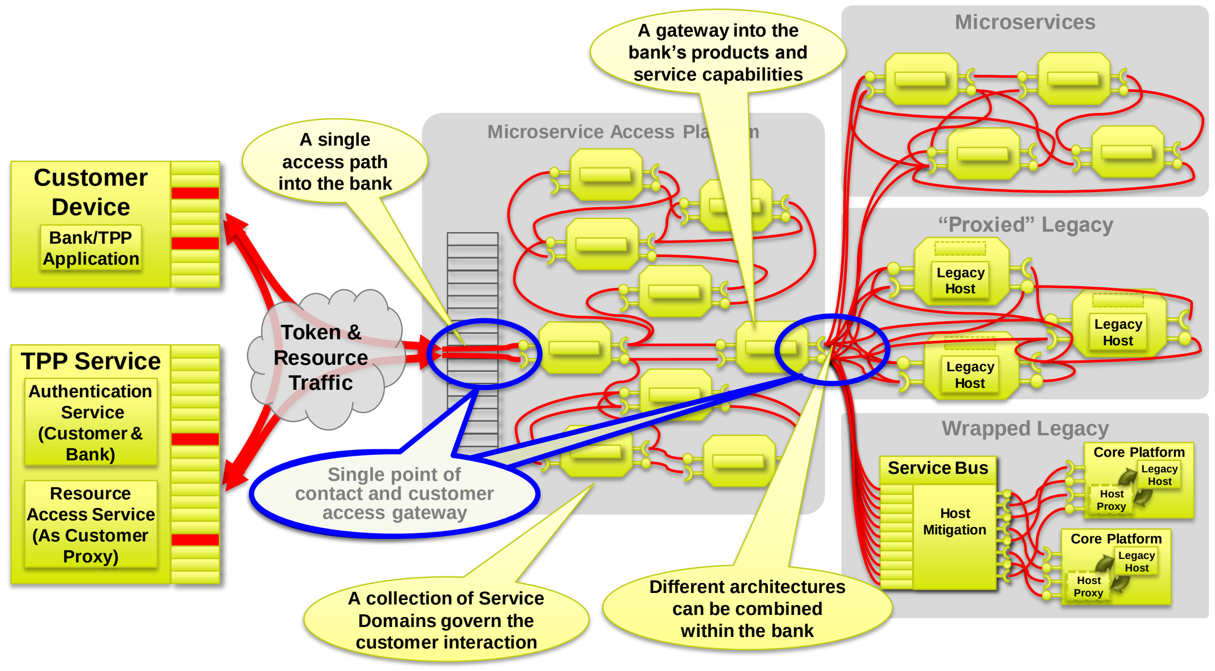

Customer Device
Bank/TPP Application
TPP Service
Authentication Service (Customer & Bank)
Resource Access Service (As Customer Proxy)
A single access path into the bank
Token & Resource Traffic
A gateway into the bank's products and service capabilities
Microservice Access Platform
Microservices
"Proxied" Legacy
Legacy Host
Wrapped Legacy
Service Bus
Host Mitigation
Core Platform
Host Proxy
Single point of contact and customer access gateway
A collection of Service Domains govern the customer interaction
Different architectures can be combined within the bank

Figure 34: Level 3 with contact points highlighted

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 discussion purposes. In the Figure 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).

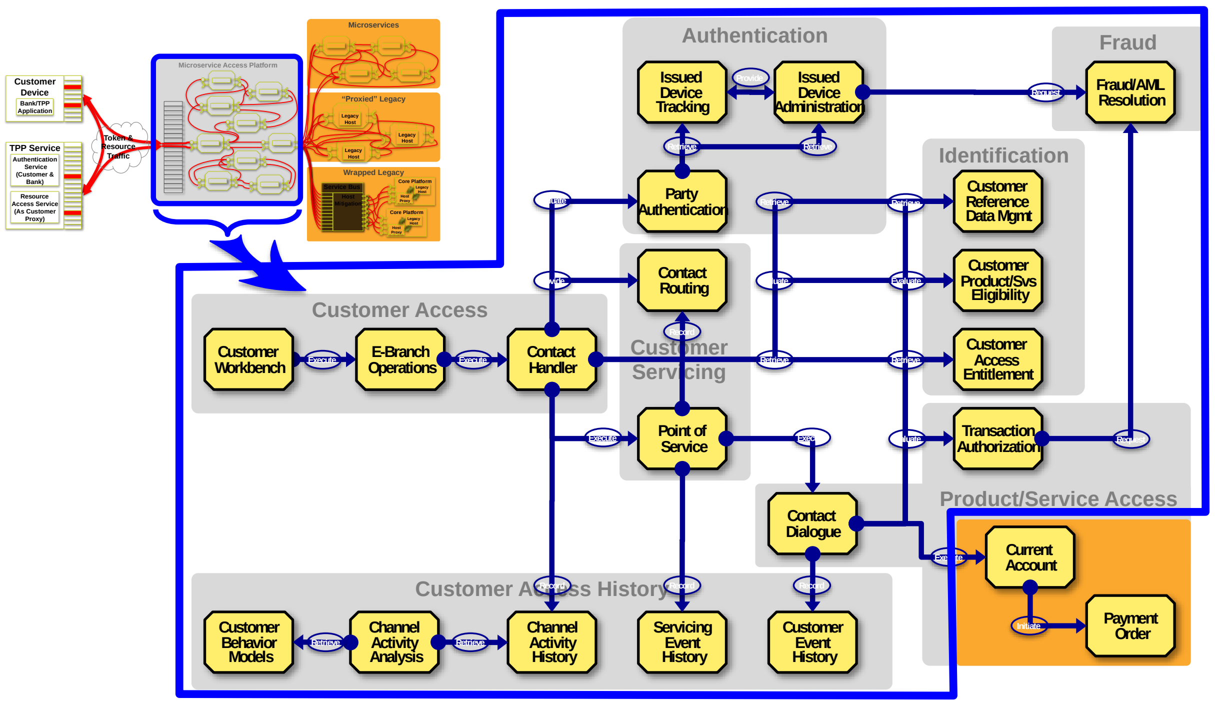


Figure 35: Level 3 - Expanded Customer Access Platform

An advantage of aligning to the BIAN standard at types 1 & 2 is that the same service API can be easily accessed through a micro-service access platform with limited re-working. The linking *Contact Dialogue* Service Domain that handles the customer interaction is the focal point for enhancements to reference available type 1 & type 2 services of the core systems as necessary.

2 - 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 using the customer access platform. 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 micro-service architecture implementation treats each Service Domain (or in cases related collections of Service Domains) as a ‘container’. The container encapsulates all function and data and all interactions are through asynchronous service exchanges. In this way the core systems can be fully componentized and by aligning to BIAN the components have standard roles/partitioning. This supports easier integration between enterprises and solution providers.

In order to fully support a micro-services solution standard definitions and approaches are required along with a connective middleware environment. The roles of the Service Domains and the purpose and payload of their service operations needs to be consistent at a fine grained level of detail.

A BIAN aligned micro-services environment has not yet been developed. It would need to address several complex considerations including:

- **Detailed data definitions** – the payload of service exchanges need to be defined to a suitable level of precision and this detail reflected in a shared business vocabulary for all service exchanges
- **Service operation service level agreements (SLA)** – the definition of the service operations need to be fully defined in terms of service make-up and operational performance and security requirements
- **Defensive operating capabilities** – every container would need to handle delayed and erroneous service operation requests and responses in a defensive manner
- **Routing capabilities** – every container would need a mechanism to discover and establish all necessary service connections for transaction execution, referential dependencies/synchronization and command and control purposes
- **State management and service triggering** – in order to ensure operational integrity, the interaction dependencies between all containers must be fully defined. Service exchanges need to be triggered by detected internal state changes
- **Exchanges must be idempotent and commutative –** meaning Service Domains needs to handle repeated/duplicated calls and the final result should

be the same regardless of the sequence of interactions that may occur when a collection of Service Domains react to an event

- **Middleware capabilities** – a connective middleware environment can provide critical service exchange support for initiating, tracking and recovering service exchanges as may be required

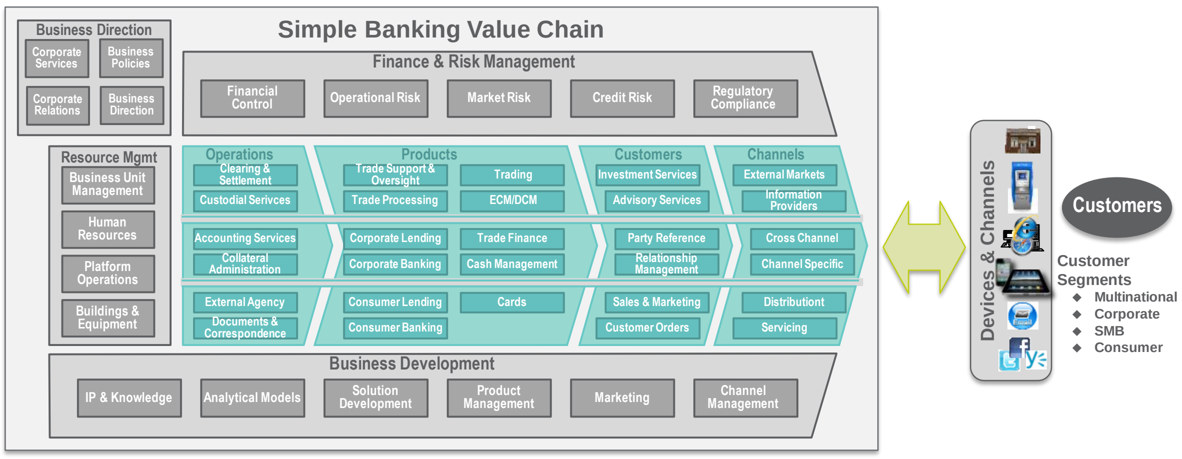
The development of a micro-service platform can be incremental. Many of the requirements can be initially addressed with manual solutions that can be replaced with increasingly automated facilities as they are developed/refined.

3 Assembling a Representative Enterprise Blueprint

The BIAN Service Landscape is a reference framework for classifying and organizing the complete collection of BIAN Service Domains. Each Service Domain is a general design of a discrete capability partition and the Service Domains collectively are intended to cover all aspects of the Banking industry. The BIAN Service Landscape is not a general model of a Bank, it is more a structured 'library' index that contains one of each of all of the possible building blocks that might be needed to build any bank.

The BIAN Service Landscape has evolved in use over the years. Its structure and a general discussion of the way Business Areas and Business Domains are used to define a Service Landscape is described in the How To Guide – Developing Content. For the creation of the enterprise blueprint a different layout of the Service Landscape has been developed. This layout better reflects some of the typical structure and flows/connections in a typical bank and so is a better format for assembling the blueprint. It does not change the nature or definition of the Service Domains it is simply a particular arrangement of the same Service Domains.

This alternative layout is called the 'value chain' view as is includes structures that align loosely to the value chain in service delivery. The Business Areas and Business Domains of the value chain layout are shown with brief descriptions in the Figure below:



Business Direction **– policy definition, direction and management oversight of all business activity**

Business Development **– covers the development of new business, products and services**

Customers **– provides an environment to maintain customer details and insights and provide customer specific control over execution across products**

Resource Management **– management administration and deployment of resources (fixed assets/staff/know-how etc.)**

Operations **– contains all functions that provide cross product/utility execution capabilities**

Channels **– handles all channel related activity including, external IP connections, controlling access and item/token distribution**

Finance & Risk Management **– financial risk/reward oversight of business activity including regulatory compliance**

Products **– covers the range of product specific execution activities, including scheduled and customer requested actions**

Figure 36: M4Bank 'Value Chain' Business Area and Business Domain layout

To underscore that the ‘value chain’ layout is simply a way to group and position the standard collection of Service Domains the following Figure shows an intermediate layout where the Service Domains from the standard Service Landscape format are first reorganized into a second matrix format with the amended Business Areas and Business Domains that are used in the value chain layout. They are then repositioned into the framework of the value chain layout itself on the far right of the Figure:

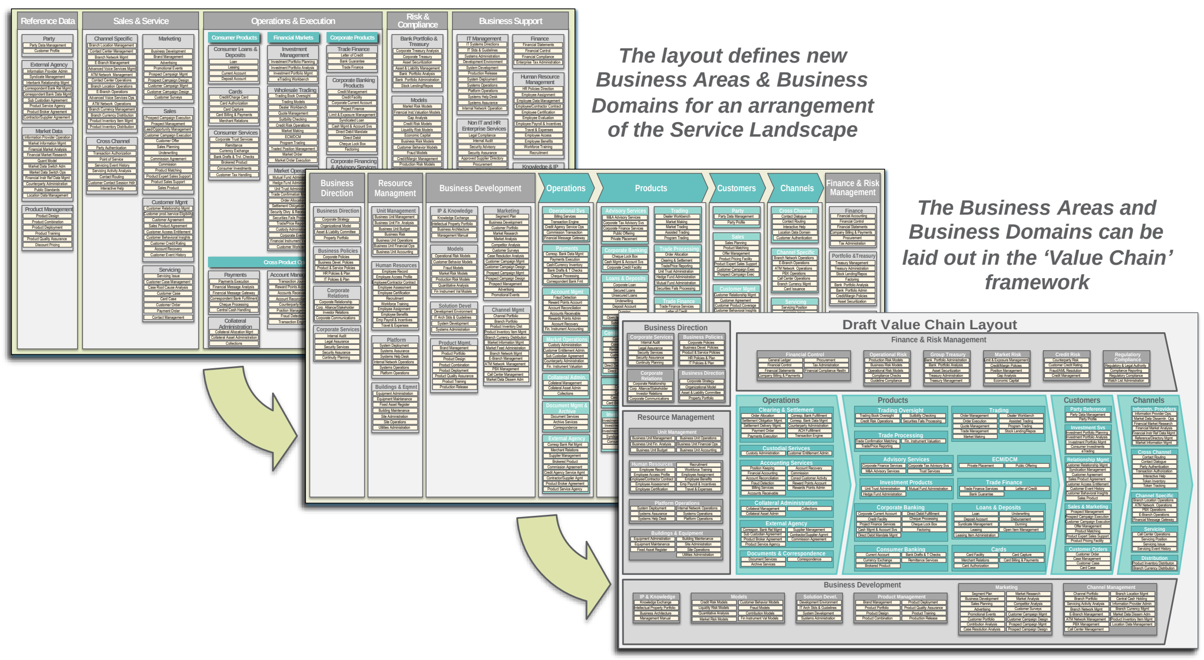

The layout defines new Business Areas & Business Domains for an arrangement of the Service Landscape
The Business Areas and Business Domains can be laid out in the ‘Value Chain’ framework
Draft Value Chain Layout
Finance & Risk Management
Operations
Products
Customers
Channels
Business Direction
Resource Management
Business Development

Figure 37: From the conventional Service Landscape to the Value Chain layout

Building the Enterprise Blueprint for a Bank

This section describes the use of the BIAN SOA Framework to develop an Enterprise Blueprint – the blueprint is often referred to as the ‘bank on a page’ view because it provides a concise representation of all of the business activities that make up the organization. The assembly of the enterprise blueprint for a particular financial institution involves three steps:

1. Select the Service Domains used by the Enterprise
2. Adapt the BIAN Service Domains as necessary
3. Assemble Service Domains in Structure Matching the Enterprise

The three steps of the approach are summarized in the Figure below. Each step is explained in more detail in the following sections.

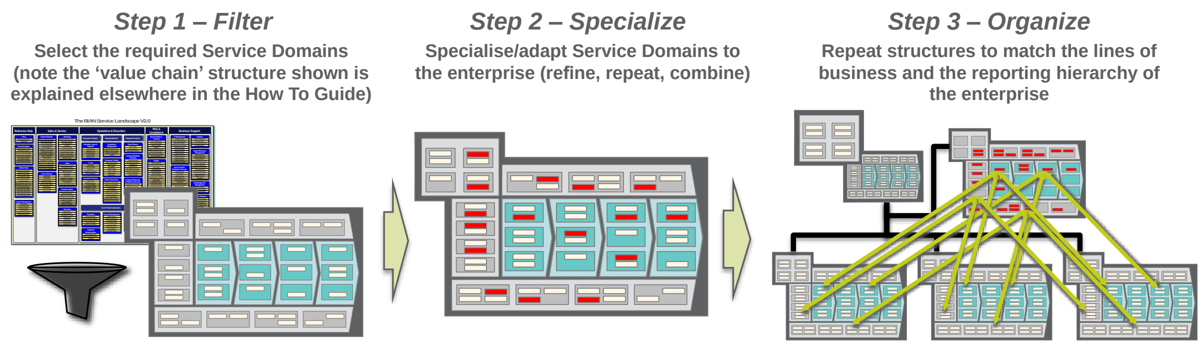

Step 1 – Filter
Select the required Service Domains (note the 'value chain' structure shown is explained elsewhere in the How To Guide)
Step 2 – Specialize
Specialise/adapt Service Domains to the enterprise (refine, repeat, combine)
Step 3 – Organize
Repeat structures to match the lines of business and the reporting hierarchy of the enterprise
The BIAN Service Landscape V2.0

Figure 38: Three steps in developing an Enterprise Blueprint

Note: the approach is described as if the model is being defined to match the desired future state of the enterprise. Depending on the specific situation the blueprint might be designed to reflect the current state and/or some intermediate states, particularly if a significant organizational restructuring is anticipated.

3.1 Select Service Domains that Match the Enterprise Activity

The first step is a simple filtering of the Service Domains of the Service Landscape to exclude any that support functions that are not required. As the BIAN landscape is intended to cover all possible activities that may be used in any bank there may be product, channel and oversight activities that don't apply to a specific organization.

In time BIAN will develop example models that can be adapted to help with the development of an enterprise model. These standard example blueprints will cover different types of financial institutions such as:

- A local/regional bank (consumer & corporate customers)
- A national bank
- A multi-national, full service bank
- An investment bank
- A hedge fund
- A private bank

3.2 Adapt the General BIAN Specifications as Necessary

The BIAN Service Domains are canonical definitions of the 'elemental' business functional capacity building blocks, meaning they define the mainstream features of a Service Domain in a way that can be consistently interpreted in different deployments. The techniques used to correctly scope and define the mainstream activities are described in detail in the BIAN How-to Guide – Developing Content.

In some specific cases the scope of a Service Domain may need to be adjusted when defining the blueprint for a bank. This is the case where there are hierarchical structures that can be different in different types of Bank. In these situations, the BIAN Service Domains can be duplicated and specialized when the BIAN definition is too coarse-grained, or combined when too fine grained. The approach is outlined below.

An example area of the landscape where this might be necessary is product fulfillment. BIAN defines generalized service domains for most main product types – an organization may well have many variants of these primary product types with their own P&L reporting and delivery features that require their own appropriately renamed Service Domains.

The general BIAN Service Domain and service operation specifications may also need to be augmented with specialized features. This may be needed for many reasons including to deal with local geo-political requirements, to support unique/differentiating business features and to deal with legacy systems behaviors.

Dealing with necessary scale variations

The right-sizing decision that BIAN strives to make may not always be unambiguous however. The BIAN standard is intended to support any type and size of financial institution. For some ‘hierarchical’ areas such as the categorization of customer types, product types and risk categories, the definition of elemental partitions and unique business context for different types and size of bank will result in different granularity for the Service Domains.

In these situations, the BIAN approach is to find a ‘mid-point’ for scoping the Service Domain within BIAN and then this design can be interpreted as necessary by a bank by either concatenating Service Domains and their associated service operations when they are too fine-grained, or duplicating and then specializing the resulting Service Domains and service operations when it is too coarse-grained as indicated in the Figure below:

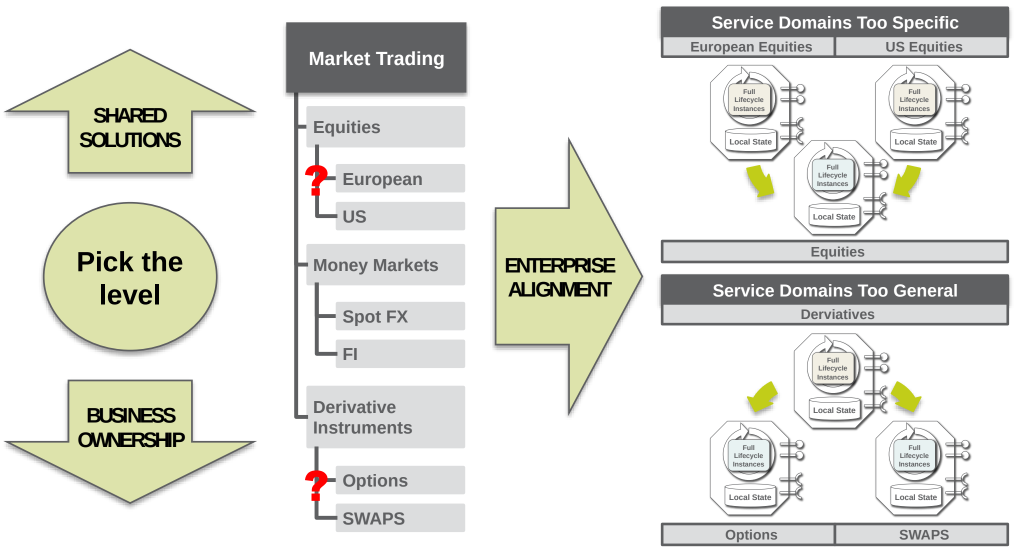


Figure 39: Combining and duplicating Service Domains

Required adjustments to the first order interactions, wireframes and business scenarios resulting from concatenating or duplicating Service Domains should be minimal and limited to the immediate vicinity of the amended Service Domains. The overall service connectivity patterns should be stable.

3.3 Assemble Service Domains in a Structure Matching the Enterprise

The final step involves assembling the selected and amended Service Domains in a structure that corresponds to the organizational layout of the enterprise. This needs to take into account the following main organizational considerations:

- *Discrete lines of business* – these can be based on geographic and/or market segment lines e.g. retail banking in a country, global trading.
- *Centralized operations* – this can be regional or global operations centers that support multiple lines of business e.g. a regional payments center, central training services.
- *Legal entity structure* – there can be head office, regional and local management units each with their own legal entity structures and obligations e.g. a global holding company with regional and local subsidiaries.

When building the enterprise blueprint a value chain ‘element’ is completed for each line of business. The content of each value chain element needs to include only those Service Domains that represent functions performed within that line of business. This involves applying the same filtering as was done at the enterprise level in the first step to every line of business to create the collection of line of business value chain elements. The Figure below shows an example of two different value chains.

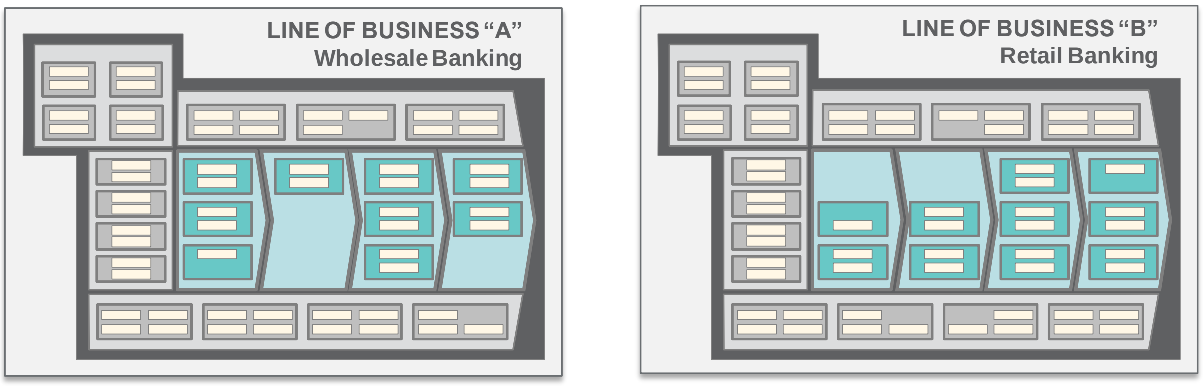


Figure 40: Two value chain elements representing different lines of business

One important advantage of the value chain layout is that is can be adapted to handle cross-line of business structures. These are needed in two specific instances in an enterprise:

1. When activities performed at the local/line of business level in two or more locations need to be consolidated at a regional level – for example for the credit risk management of a multinational customer relationship.

2. When a shared/centralized service is offered to two or more locations – for example centralized payments services or trade clearing and settlement processing.

In these situations, an additional value chain 'element' is completed containing only those service domains that handle those common functions that need to be coordinated and any shared/centralized functions. The way common and shared functions for two lines of business are linked to a cross line of business element is shown in the Figure below:

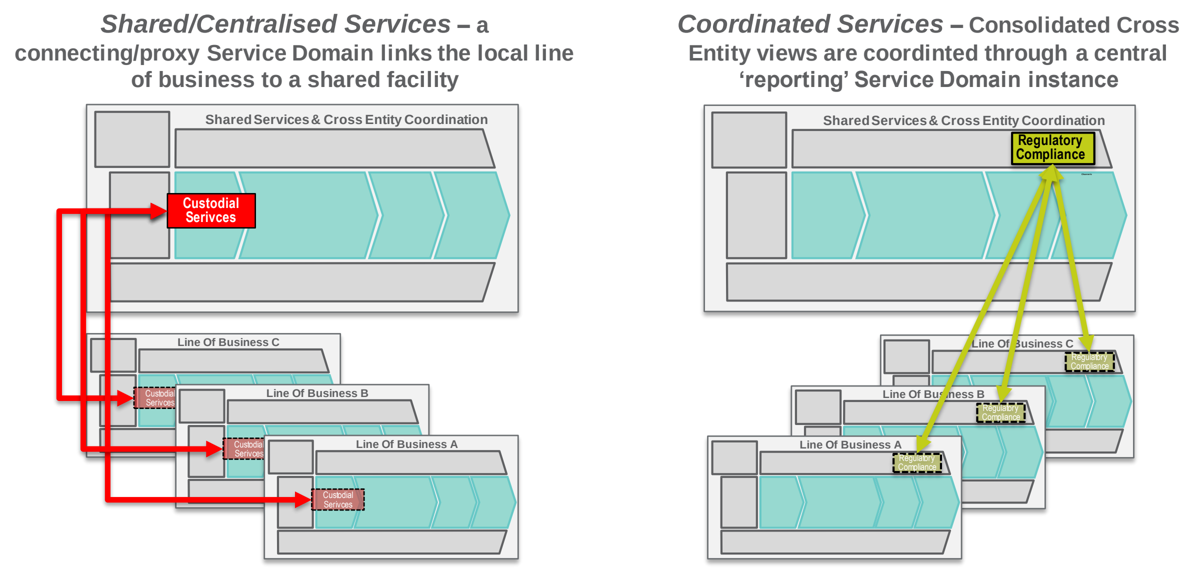

Shared/Centralised Services – a connecting/proxy Service Domain links the local line of business to a shared facility
Shared Services & Cross Entity Coordination
Custodial Serivces
Line Of Business C
Custodial Services
Line Of Business B
Custodial Services
Line Of Business A
Custodial Services
Coordinated Services – Consolidated Cross Entity views are coordinted through a central 'reporting' Service Domain instance
Shared Services & Cross Entity Coordination
Regulatory Compliance
Line Of Business C
Regulatory Compliance
Line Of Business B
Regulatory Compliance
Line Of Business A
Regulatory Compliance

Figure 41: Two lines of business connected to a regional operation

The final aspect of the enterprise blueprint reflects the organization's legal entity structure. As the value chain model includes all of the business activities that make up any one operating unit, it can be used as a management dashboard to capture the reporting for all entities within that unit or 'element'. As the same value chain layout is used for a line of business or regional reporting center, the management region of the value chain can be used to support a reporting hierarchy from the individual lines of business through any regional units into the central head office management function.

This combined structure is shown in the Figure below:

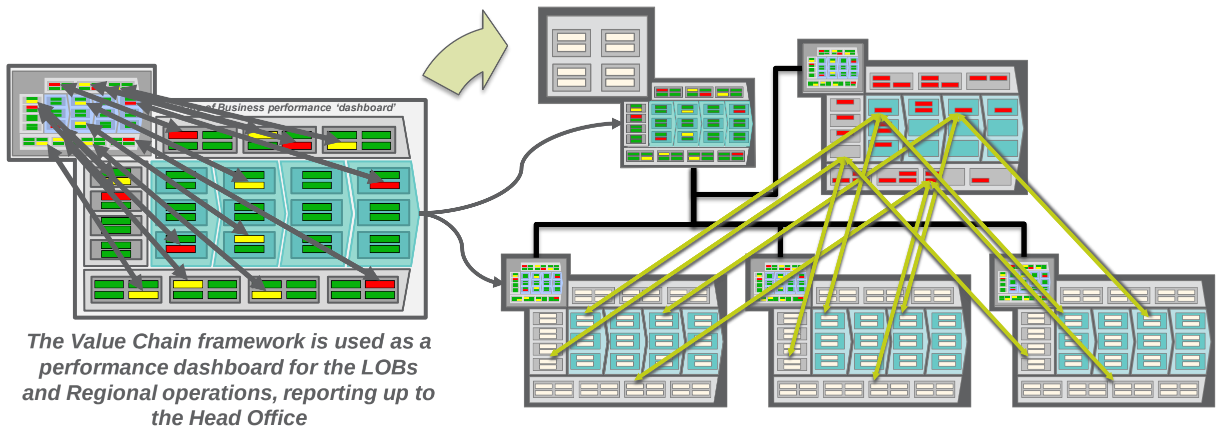

Business performance 'dashboard'
The Value Chain framework is used as a performance dashboard for the LOBs and Regional operations, reporting up to the Head Office

Figure 42: M4Bank with local units, regional and head office reporting

The line of business structure, make-up of any regional oversight or shared operations, and the legal entity reporting structure for the simple example above can clearly be adapted to represent the corresponding structure of any bank to produce their own specific enterprise blueprint.

Matching the Enterprise Segmentation Approach

Different banks adopt different approaches to market segmentation, matching types of products and services to specific types of customer. A particular issue for banks interpreting the BIAN model is mapping their segmentation to the products and customer types reflected in the BIAN Service Landscape. To provide the greatest flexibility BIAN seeks to define generic product types independent of the target customer type. These general products can then be aligned and if necessary specialized to support the target customer groups. In this way a generic product Service Domain may be specialized to create multiple product fulfillment variants to match a specific Bank's segmentation.

This concept is shown in the next Figure where generic product types are listed in the vertical dimension and generic customer categories across the top. Examples of specialized product variants can be seen in the matrix. A typical segmentation footprint has then been overlain as an example show the overlaps. BIAN is currently reviewing the way product hierarchies and coverage is handled in the Service Landscape and revisions may be made in future releases

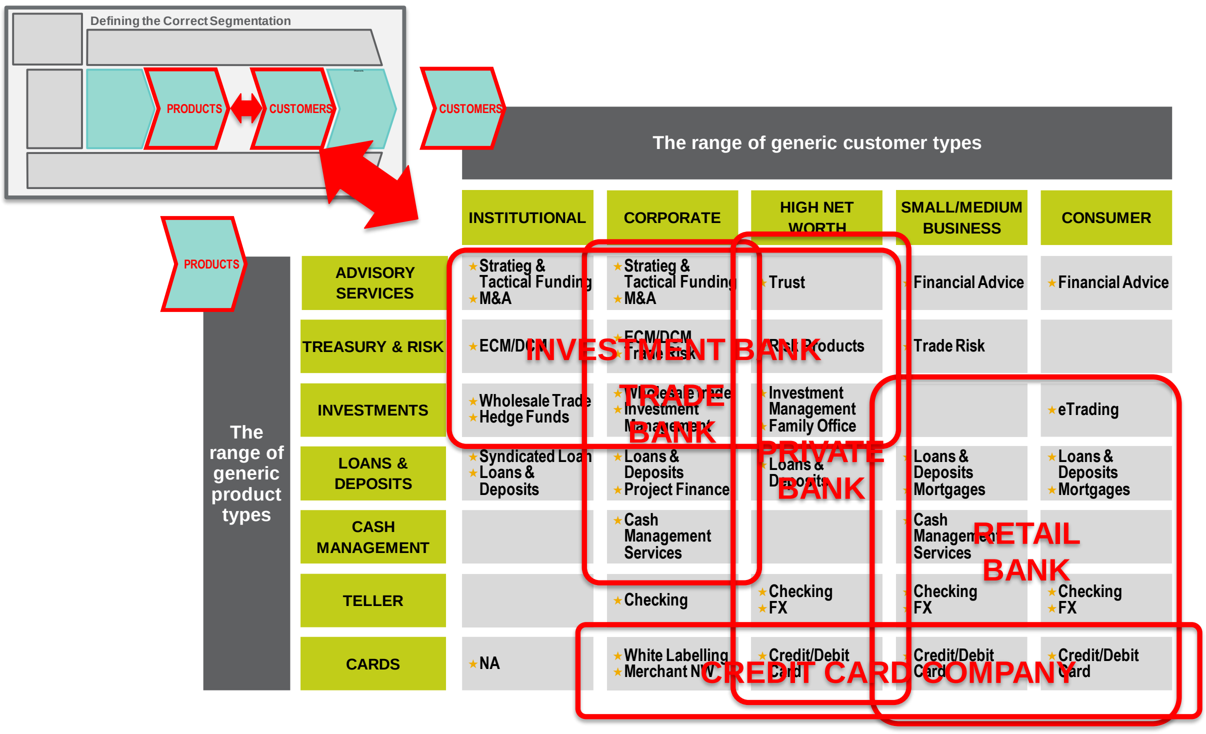


Figure 43: Mapping product and customer types to segmentation views

4 An Enterprise Blueprint is a Framework for Analysis

An enterprise blueprint built using BIAN Service Domain partitions is a particularly good framework for management analysis and planning for a number of reasons:

- *Stable over time* - the roles of the discrete business functional capacities represented by Service Domain partitions do not vary over time. The way they might execute and the patterns of collaboration will change as new practices and solutions evolve, but their specific purpose and hence their position in the blueprint is unlikely to change. As a result, a model built using these elements will itself be highly enduring.

- *Concise, implementation independent view* - beyond the broad-brush scope of business areas, the blueprint does not presume any specific organizational or technical approach. It can also be interpreted both by business and technical architects consistently bridging a design gap that often exists with other (incompatible) business and technology enterprise modeling approaches.

- *Suited for overlays/attribution* – the elements that make up the model can be associated with resources and current or target performance measures to support a wide range of business performance assessments.

- *High-level design* – the blueprint can also act as the top level design of underlying organizational, procedural and information systems solutions providing a planning framework for targeting investment and solution development.

The enterprise blueprint provides a framework that can support a wide range of planning and analysis functions. Earlier in this section a list of projects that might make use of an enterprise blueprint were listed:

- Application Portfolio Rationalization – the target requirements and current application capabilities are overlain and shortfalls identified.
- *Mergers & Acquisitions* – the combined application portfolios of the merged organizations can be mapped and based on target capabilities the optimum systems selected for the merged organizations.
- Etc…

These projects and many other similar types of initiative use the enterprise blueprint in a similar way: It can be used to associate required features and properties with parts of the business aligned to Service Domains. Note the approaches described below may also be applied to coarser grained clusters of Service Domains within a Business Domains as appropriate. The descriptions that follow reference Service Domains for brevity.

Performance measurements can be set and tracked for these parts of the business and resources, such as staff and systems, can be mapped against the parts of the business to assess coverage. The following sections describe these types of uses in more detail.

4.1 The BIAN Specifications can be augmented

The BIAN Service Domain and service operations can be augmented to provide as much detail as may be required. Already discussed is the approach for expanding the feature checklists to include the prevailing or target functionality for the Service Domain. As described earlier – and many additional templates will be considered. Also specialization based on geopolitical scale and sophistication/maturity

The Service Domains and service operations can also be matched to industry standard and proprietary specifications. BIAN is currently developing its own business object model that is informed by the industry standard ISO20022 model. Links to other standards such as OMG/FIBO are under consideration.

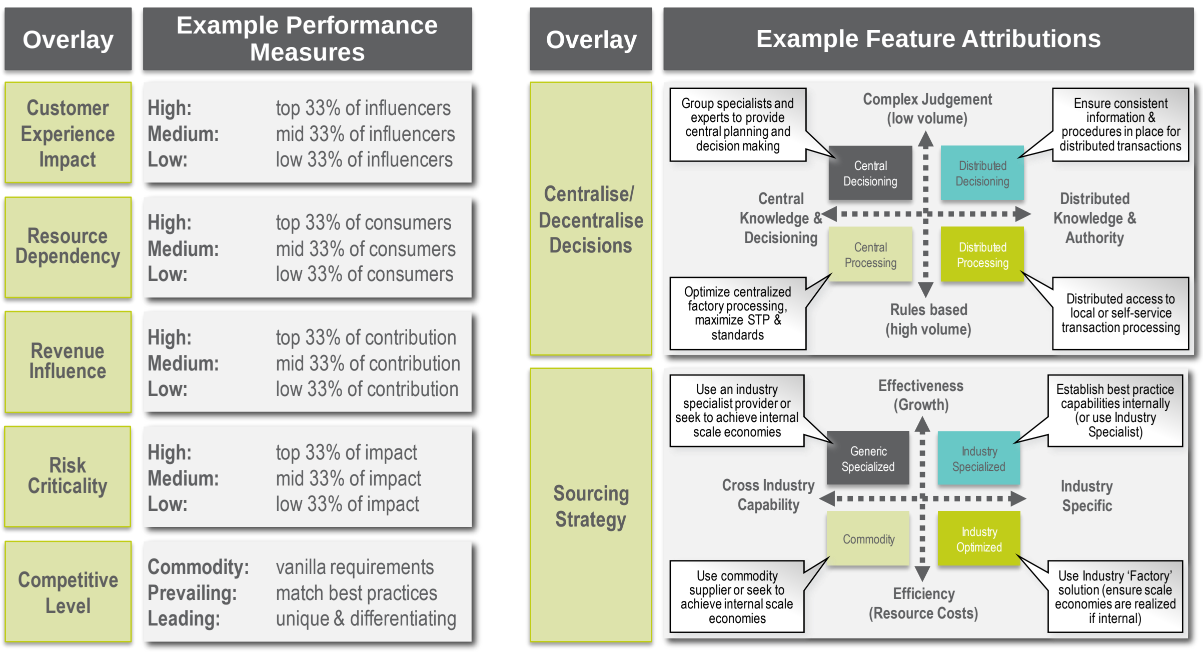
Various initiatives to map BIAN to other standards are described in white papers that can be found at BIAN.org

Feature attribution

In addition to adding functional detail to the model, a wide range of 'attributions' can be associated with a Service Domain. These features can relate to the supporting system properties or wider operational and business considerations. The features usually define target state requirements, adding additional insights to the target state functionality.

For some attributions the approach to assigning a value or rating is simple direct assessment. For others there can be a simple associated technique to determine which Service Domains make a specific contribution and to compare Service Domains against one another to get a comparative ranking. It is not BIAN's role to define these techniques, but by providing examples it is intended that members and others applying the BIAN standard will develop their own approaches to leverage the standard.

The following table includes a number of attributions that can be applied to Service Domains as examples. The range of possible attributions is unlimited and can be developed to support different types of planning and assessment.



Overlay	Example Performance Measures	
Customer Experience Impact	High: Medium: Low:	top 33% of influencers mid 33% of influencers low 33% of influencers
Resource Dependency	High: Medium: Low:	top 33% of consumers mid 33% of consumers low 33% of consumers
Revenue Influence	High: Medium: Low:	top 33% of contribution mid 33% of contribution low 33% of contribution
Risk Criticality	High: Medium: Low:	top 33% of impact mid 33% of impact low 33% of impact
Competitive Level	Commodity: Prevailing: Leading:	vanilla requirements match best practices unique & differentiating

Overlay
Example Feature Attributions
Centralise/ Decentralise Decisions
Group specialists and experts to provide central planning and decision making
Complex Judgement (low volume)
Ensure consistent information & procedures in place for distributed transactions
Central Decisioning
Distributed Decisioning
Central Knowledge & Decisioning
Distributed Knowledge & Authority
Central Processing
Distributed Processing
Optimize centralized factory processing, maximize STP & standards
Rules based (high volume)
Distributed access to local or self-service transaction processing
Sourcing Strategy
Use an industry specialist provider or seek to achieve internal scale economies
Effectiveness (Growth)
Establish best practice capabilities internally (or use Industry Specialist)
Generic Specialized
Industry Specialized
Cross Industry Capability
Industry Specific
Commodity
Industry Optimized
Use commodity supplier or seek to achieve internal scale economies
Efficiency (Resource Costs)
Use Industry 'Factory' solution (ensure scale economies are realized if internal)

Figure 44: Example attributions

As can be seen in the table some attributions are simple rankings that can be applied across the blueprint quite easily, some others are a little more involved. Two examples are explained in a little more detail.

In the specific case of *strategic intensity,* the Service Domains receive a relative rating at three levels:

- *Commodity* – the performance of the Service Domain is commodity in nature – there are no optional features that provide differentiation – all participants needs to perform to the same basic level.
- *Prevailing Practices* – there is a range of operating approaches and the organization needs to aspire to matching prevailing 'good' practices so that it is not compared negatively with its peer competitors.
- *Differentiated* – there is some aspect of the operation that is unique or distinct from the competitors that provides a differentiating advantage in the market (faster/better/cheaper).

The assignment of competitive intensity to the Service Domains makes use of an analysis of the strategic intent of the enterprise. Give the strategic goals and objectives it is necessary to identify which Service Domains contribute directly to the achievement of those goals and how (Business Scenarios can be used to achieve this mapping if it is not obvious).

Two other attributions for sourcing and centralization use a quadrant ranking technique to classify a Service Domain. The quadrants for a centralization assessment are shown in the next Figure:

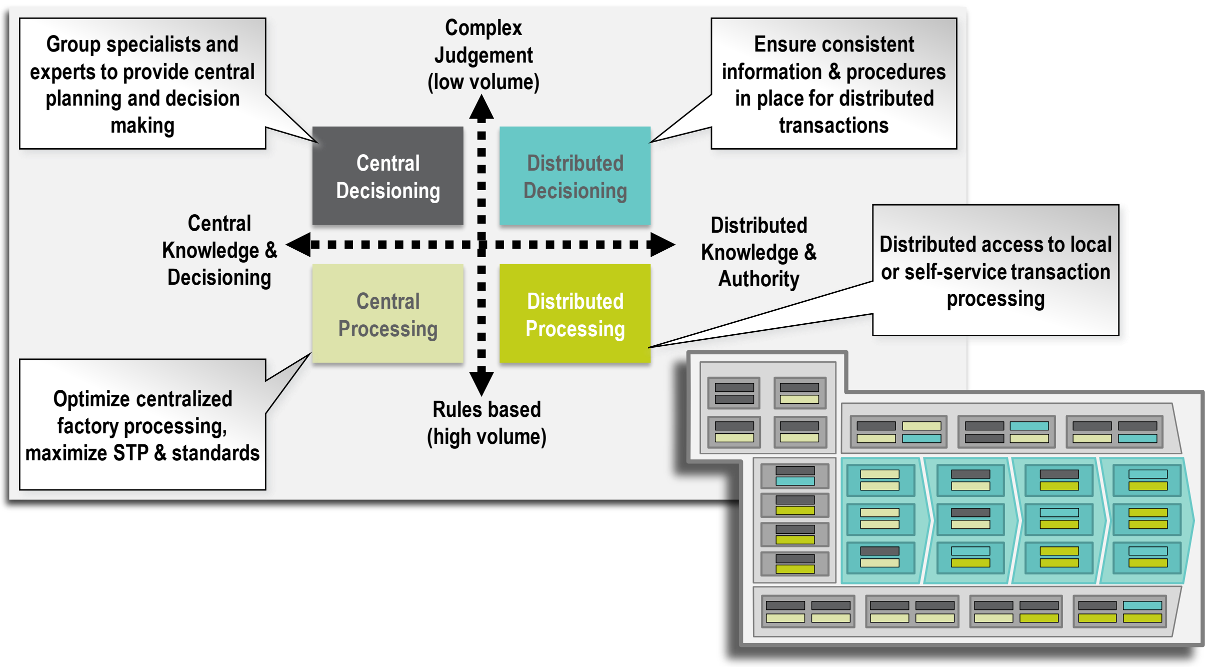
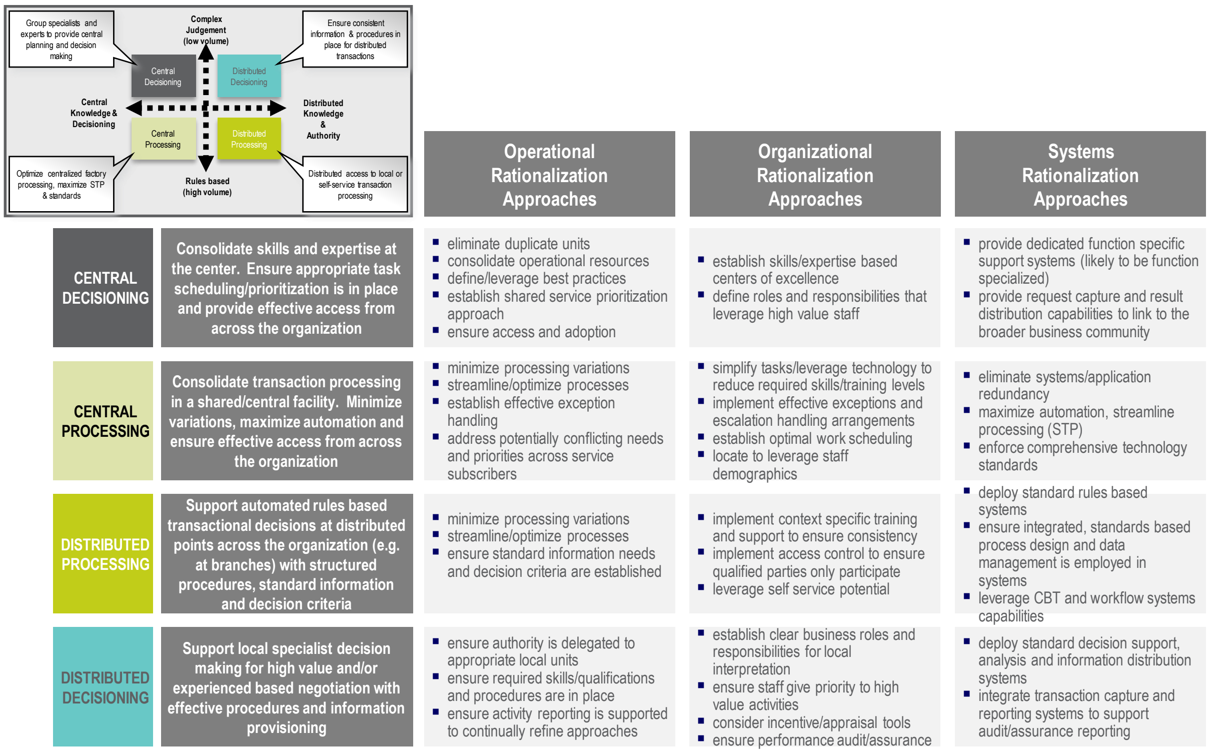

Group specialists and experts to provide central planning and decision making
Complex Judgement (low volume)
Ensure consistent information & procedures in place for distributed transactions
Central Decisioning
Distributed Decisioning
Central Knowledge & Decisioning
Distributed Knowledge & Authority
Distributed access to local or self-service transaction processing
Central Processing
Distributed Processing
Optimize centralized factory processing, maximize STP & standards
Rules based (high volume)

Figure 45: Attribution quadrant with an attributed value chain element

The Service Domains are assigned to a quadrant based on their business and operational characteristics. Based on this classification strategies and techniques can be associated with the organizational, operational and systems related approaches for the Service Domain, per the next Figure:


Group specialists and experts to provide central planning and decision making
Complex Judgement (low volume)
Ensure consistent information & procedures in place for distributed transactions
Central Decisioning
Distributed Decisioning
Central Knowledge & Decisioning
Distributed Knowledge & Authority
Central Processing
Distributed Processing
Optimize centralized factory processing, maximize STP & standards
Rules based (high volume)
Distributed access to local or self-service transaction processing

		Operational Rationalization Approaches	Organizational Rationalization Approaches	Systems Rationalization Approaches
CENTRAL DECISIONING	Consolidate skills and expertise at the center. Ensure appropriate task scheduling/prioritization is in place and provide effective access from across the organization	▪ eliminate duplicate units ▪ consolidate operational resources ▪ define/leverage best practices ▪ establish shared service prioritization approach ▪ ensure access and adoption	▪ establish skills/expertise based centers of excellence ▪ define roles and responsibilities that leverage high value staff	▪ provide dedicated function specific support systems (likely to be function specialized) ▪ provide request capture and result distribution capabilities to link to the broader business community
CENTRAL PROCESSING	Consolidate transaction processing in a shared/central facility. Minimize variations, maximize automation and ensure effective access from across the organization	▪ minimize processing variations ▪ streamline/optimize processes ▪ establish effective exception handling ▪ address potentially conflicting needs and priorities across service subscribers	▪ simplify tasks/leverage technology to reduce required skills/training levels ▪ implement effective exceptions and escalation handling arrangements ▪ establish optimal work scheduling ▪ locate to leverage staff demographics	▪ eliminate systems/application redundancy ▪ maximize automation, streamline processing (STP) ▪ enforce comprehensive technology standards
DISTRIBUTED PROCESSING	Support automated rules based transactional decisions at distributed points across the organization (e.g. at branches) with structured procedures, standard information and decision criteria	▪ minimize processing variations ▪ streamline/optimize processes ▪ ensure standard information needs and decision criteria are established	▪ implement context specific training and support to ensure consistency ▪ implement access control to ensure qualified parties only participate ▪ leverage self service potential	▪ deploy standard rules based systems ▪ ensure integrated, standards based process design and data management is employed in systems ▪ leverage CBT and workflow systems capabilities
DISTRIBUTED DECISIONING	Support local specialist decision making for high value and/or experienced based negotiation with effective procedures and information provisioning	▪ ensure authority is delegated to appropriate local units ▪ ensure required skills/qualifications and procedures are in place ▪ ensure activity reporting is supported to continually refine approaches	▪ establish clear business roles and responsibilities for local interpretation ▪ ensure staff give priority to high value activities ▪ consider incentive/appraisal tools ▪ ensure performance audit/assurance	▪ deploy standard decision support, analysis and information distribution systems ▪ integrate transaction capture and reporting systems to support audit/assurance reporting

Figure 46: Example approaches associated with an attribution

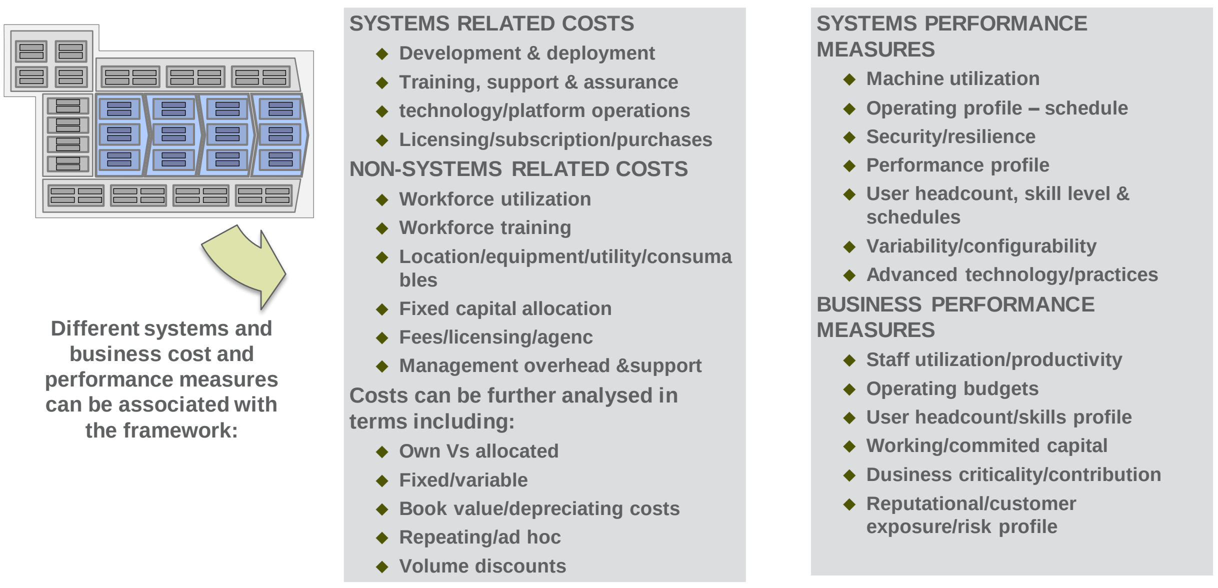
As noted a wide range of attribution types and attribution techniques and guidelines can be developed and applied to the enterprise blueprint for an organization to help clarify the requirements/intent for the Service Domain aligned business functions that make up the organization.

These attributions can be referenced in different combinations to inform investment decisions and help align subsequent investment projects.

4.2 Track Business and Technical Performance

The enterprise blueprint can also be used as a management dashboard. Because it represents a stable view of the discrete functions that make up the organization, a wide range of performance measures can be associated with these individual 'units' and these used to set targets and track performance to plan.

The measures can address systems performance or can target wider operational and organizational measures. As with attribution, the list of possible measurements is unlimited, some examples include the elements shown in the next Figure:


Different systems and business cost and performance measures can be associated with the framework:

SYSTEMS RELATED COSTS
- Development & deployment
- Training, support & assurance
- technology/platform operations
- Licensing/subscription/purchases

NON-SYSTEMS RELATED COSTS
- Workforce utilization
- Workforce training
- Location/equipment/utility/consumables
- Fixed capital allocation
- Fees/licensing/agenc
- Management overhead &support

Costs can be further analysed in terms including:
- Own Vs allocated
- Fixed/variable
- Book value/depreciating costs
- Repeating/ad hoc
- Volume discounts

SYSTEMS PERFORMANCE MEASURES
- Machine utilization
- Operating profile – schedule
- Security/resilience
- Performance profile
- User headcount, skill level & schedules
- Variability/configurability
- Advanced technology/practices

BUSINESS PERFORMANCE MEASURES
- Staff utilization/productivity
- Operating budgets
- User headcount/skills profile
- Working/commited capital
- Dusiness criticality/contribution
- Reputational/customer exposure/risk profile

Figure 47: Systems and operational cost & performance measures

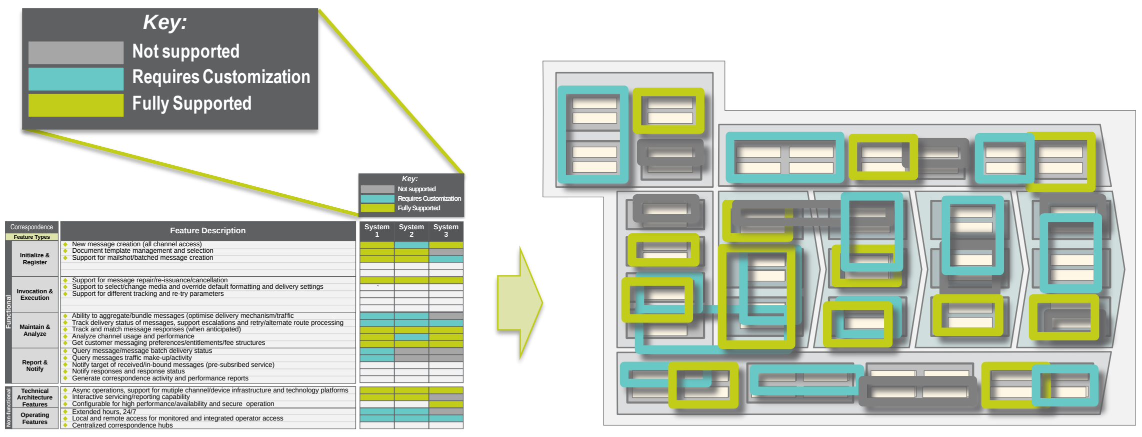
The enterprise blueprint itself can be used to format a management dashboard, with values/color coding related to the Service Domains to provide a succinct visual representation of the organizations operating state against the plan.

4.3 Overlay Resources to Identify Shortfalls

A final type of use of the enterprise blueprint is to overlay resources to identify shortfalls in the level of support. This approach has been used earlier in this document where current systems and commercial offering were compared to the Service Domains for a point solution.

The same mapping approach, optionally using the finer grained feature lists for Service Domains can be used across regions or the entire blueprint. Resources that can be mapped include systems/applications, personnel and utilities such as technology platforms, building and equipment facilities.

As described earlier, resource mappings can be used to highlight shortfalls such as gaps, duplication and misaligned resources as summarized in the next Figure:



Key: ▨ Not supported; 🟦 Requires Customization; 🟩 Fully Supported

Correspondence		Feature Description	System 1	System 2	System 3
Feature Types					
Functional	Initialize & Register	New message creation (all channel access)	🟩	🟦	🟩
		Document template management and selection	🟩	🟩	▨
		Support for mailshot/batched message creation	🟩	🟩	🟦
	Invocation & Execution	Support for message repair/re-issuance/cancellation	🟩	🟩	🟩
		Support to select/change media and override default formatting and delivery settings			
		Support for different tracking and re-try parameters			
	Maintain & Analyze	Ability to aggregate/bundle messages (optimise delivery mechanism/traffic	🟦	🟦	▨
		Track delivery status of messages, support escalations and retry/alternate route processing	🟦	🟦	🟦
		Track and match message responses (when anticipated)	🟩	🟩	🟩
		Analyze channel usage and performance	🟩	🟦	🟩
		Get customer messaging preferences/entitlements/fee structures	🟩	🟩	🟩
	Report & Notify	Query message/message batch delivery status	🟦	▨	▨
		Query messages traffic make-up/activity	🟦	▨	▨
		Notify target of received/in-bound messages (pre-subsribed service)			
		Notify responses and response status			
		Generate correspondence activity and performance reports			
Non-functional	Technical Architecture Features	Async operations, support for mutiple channel/device infrastructure and technology platforms	🟩	🟩	🟩
		Interactive servicing/reporting capability	🟩	🟩	▨
		Configurable for high performance/availability and secure operation			🟩
	Operating Features	Extended hours, 24/7	🟦	🟦	▨
		Local and remote access for monitored and integrated operator access	🟦	🟦	🟦
		Centralized correspondence hubs			

Service Domain level systems mapping ***Shapes show application functional footprints***

Figure 48: Overlay of systems on an enterprise blueprint revealing shortfalls

A final observation, having described the definition of the enterprise blueprint and the different types of analysis it can support. The elements of the blueprint align to Service Domains and as described earlier in this document those same Service Domains can be formally mapped to the supporting systems. As a result, the blueprint provides a representation of the business that can capture business needs and priorities and relate this to the underlying supporting systems as highlighted in the next figure:

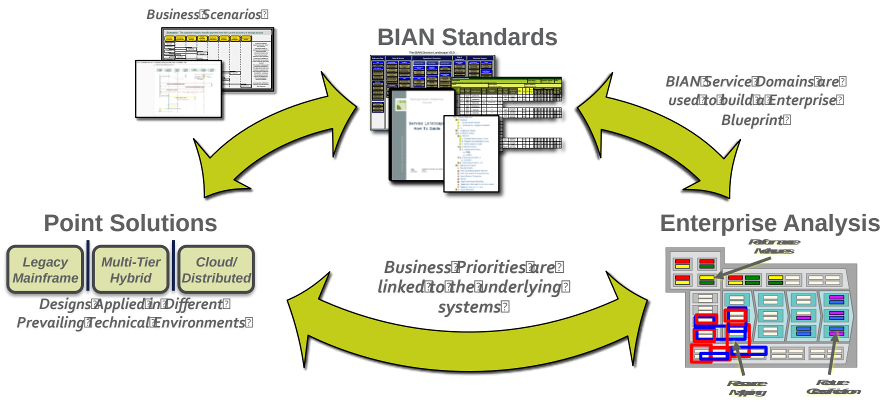

Business Scenarios
BIAN Standards
BIAN Service Domains are used to build a Enterprise Blueprint
Point Solutions
Legacy Mainframe
Multi-Tier Hybrid
Cloud/ Distributed
Designs Applied in Different Prevailing Technical Environments
Business Priorities are linked to the underlying systems
Enterprise Analysis

Figure 49: BIAN designs applied to point & enterprise solution

5 Conclusion

This document covering the application of BIAN standards is intended to provide guidance to anyone deploying solutions using the BIAN standard. It summarizes the anticipated benefits from adopting the BIAN type of service oriented architecture and explains how the BIAN designs can be augmented to provide more detailed requirements definitions for solution implementation.

This guide also addresses how these implementation level requirements may be developed for legacy repurposing and package selection and how the interpretation of the service based partitions applies differently in many prevailing technical environments. In particular, an outline of how the BIAN standard can be used as a high level specification for API development and where appropriate the adoption of a micro-service architecture. Note that a dedicated How To Guide is also available addressing the use of BIAN for API solutions in more detail.

The ideas and approaches set out are constantly being reviewed and refined in practice. BIAN will endeavor to maintain these guidelines, refining the descriptions and adding new techniques and approaches based on experience and feedback from its membership and the industry in general.