Banking Industry
Architecture Network

BIAN
Semantic API Practitioner's Guide

Organization

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

Status			
Status	**Date**	**Actor**	**Comment / Reference**
DRAFT	December 2019	BIAN Membership reviews	Revised June 2020
Approved		BIAN Architectural Committee	

Version		
No	**Comment / Reference**	**Date**
8.1	First version (for limited release)	July 2020

Copyright

Table of Contents

Table of Diagrams

1 – Contents & Introduction

Contents

This guide describes the techniques for systems technical leads and architects to interpret and apply the BIAN Semantic APIs. It covers both legacy system enhancements and new system development. It is set out as follows:

1. **Introduction** – why use BIAN semantic APIs for development?

2. **Some Key Design Considerations** – explains the key design properties of the BIAN standard that technical leads and architects need to be aware of

3. **The BIAN Specification** – describes the BIAN design artifacts available for technical leads and architects:

    a. The BIAN Service Landscape
    b. BIAN Service Domain Specifications
        i. Functional Patterns
        ii. Asset Types & Right-sizing Service Domains
        iii. Control Records
        iv. Behavior Qualifiers
        v. Service Operations and Action Terms
        vi. Service Domain First Order Connections
        vii. Service Domain Information Profile
        viii. The Figure “8” Diagram
    c. BIAN Business Scenarios
    d. BIAN Wireframes
    e. BIAN Semantic APIs (The BIAN Semantic API Portal)
    f. Service Domain Event Triggering (proposed future BIAN extension)

4. **Applying BIAN designs in different implementation contexts –** Development covers both “back-office” transaction systems and interactive and decision oriented “front-office” customer facing systems. Approaches combine integrating with conventional process based designs and developing more advanced container based architectures:

    PART 1 – Key Properties of Component Design – clarifying the key development implications of adopting a component architecture
    PART 2 - Adding Detail to the BIAN Service Domain Specifications – extending the BIAN semantic conceptual designs for implementation
    PART 3 - Implementation Approaches – detailing specific approaches to physical application designs that leverage the component model

Introduction

The financial services industry is experiencing intense pressure as the status quo is challenged by: sweeping regulatory changes; the proliferation of advanced technologies; and, FinTechs testing traditional banking models. Finally, it seems many banks are ready to confront the severe limitations of their aging systems. At the same time, they seek to position themselves to compete with new and emerging banking practices. Information technology represents both a significant constraint and a key enabler for banks during this period of radical transformation.

But there is a fundamental technology challenge: most legacy systems and even many more recent system developments have been designed to streamline and automate highly structured banking processes. Over time the cycle of incremental process automation has resulted in increasingly fragmented and overlapping application portfolios as new process oriented solutions have simply been superimposed on existing processing facilities.

To break this cycle both to repurpose legacy systems and integrate new advanced solutions banks need to adopt a very different approach: one based on a ‘componentized’ model of the banking business. This component view of banking defines discrete functional building blocks that can be flexibly assembled to support the business.

Drivers of industry componentization

When an industry achieves componentization it’s participants can specialize and develop more advanced individual components that can then be combined to collectively deliver more sophisticated products and services overall. A componentized industry is also better able to exploit new technologies and approaches as the associated change is typically localized, impacting only one or a few components without destabilizing others.

As some simple industry comparisons below show, the auto industry is a good example of a highly componentized marketplace where most vehicles combine parts from multiple specialist suppliers allowing the industry to offer ever more sophisticated products. Perhaps a less obvious example is the film and entertainment industry. With its clearly defined roles (e.g. actors, writers, producers, directors) a project team can quickly be assembled to support the production of a single film. Furthermore, its recent rapid adoption of computer-generated imagery (CGI) is a good example of the film industry’s resilience to change. For the finance industry however, defining its own component ‘blueprint’ has proven elusive.

A common industry 'blueprint' allows participants to specialise. Collectivey they can assemble more sophisticated solutions.

COMPONENT BLUEPRINT For Manufacturing: The Auto Industry…

…defines the different parts that are assembled in a car

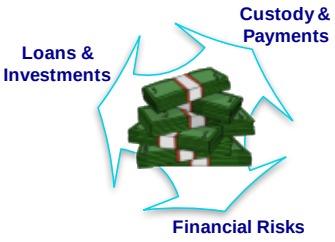
Example Disruptive Change: Electronics

COMPONENT BLUEPRINT For Media: The Film Industry…

…agrees specialist roles involved in making a film

Example Disruptive Change: Digital Media & CGI

COMPONENT BLUEPRINT For Financial Services: Banking???



…what model defines the financial services 'components'?

Example Disruptive Change: FinTechs & Technology

BIAN is defining the banking component model

Figure 1 - Comparing Componentized Industries

The BIAN standard defines a component business blueprint for banking. It has been developed specifically to address the problem of application portfolio complexity, enabling banks to progressively componentize their business operations. It adopts a novel business model view to identify the standard business functional components (the BIAN "Service Domains").

In recent years BIAN has extended the detail of the Service Domains' service operation specifications. These extensions provide a semantic definition of the underlying exchanges that can be interpreted as the high-level application programming interface (API) requirements that connect these components together.

This guide outlines the BIAN standard and defines the particular approach for systems designers and builders to apply BIAN semantic APIs to wrap/re-purpose legacy systems and to integrate new container based 'micro/macro-service' solutions into the bank's application portfolio. It considers the approach needed for restructuring the back end transaction-processing systems and also addresses the far more interactive workforce and customer facing systems that cover activities such customer servicing, new business development, risk management and product delivery and distribution.

This guide describes the foundational BIAN architectural design principles and techniques to the level necessary for technical leads and architects to correctly interpret and apply the BIAN standard. A more thorough explanation of the BIAN approach is

available in architectural training and presentation materials available on the BIAN website and through BIAN general training and certification services.

2 – Some Key BIAN Design Considerations

The BIAN Service Domains and associated semantic APIs define the mainstream business requirements at a high level that must then be extended to develop comprehensive implementation level specifications. It is not necessary for technical leads and architects to become overly proficient in the underlying BIAN architectural methods to be able to apply the BIAN designs but familiarity with the core concepts can help with the correct interpretation and application of the BIAN model.

This Section 2 covers the following design considerations/explanations:

1. Why bother with BIAN – what are the main benefits?
2. BIAN uses an Asset Leverage model view – what is this?
3. Component Vs process business model views – an example for comparison
4. The component building block – outlining key features of a BIAN Service Domain

2.1 Why bother with BIAN?

If the BIAN standard only provides high-level semantic business descriptions that require quite extensive effort to extend them to implementation level detail, what is the value BIAN provides to development? Doesn't it simply add another step in an already complicated design and development process?

Technical leads and architects can think of a Service Domain as the conceptual design for a major functional module, a handful of which might be found in any large production application. The Service Domains have been specified to have architectural properties that can have a profound impact on aligned systems solutions:

1. **Discrete/non-overlapping and elemental business functions support application containerization and operational re-use** – each Service Domain matches a discrete (unique/non-overlapping) business function that is elemental in its role (i.e. is only assignable in its entirety). A Service Domain is also specified to be responsible for handling the full lifecycle of its business function every time it is undertaken. As a result, a Service Domain encapsulates its business information and logic. The Service Domain can be implemented as a discrete service container with all external information exchanges handled through service operations.

Less obvious is the potential for operational re-use. When business activity is modeled using the BIAN approach the work tasks that might otherwise be hard-wired into a specific automated process are replaced by discrete, specialized, service-enabled business functions that can be re-used in any applicable business context leading to very high degrees of operational capability re-use.

2. **Stable over time supporting incremental development and adoption** – a Service Domain has a business role or purpose that is defined independently of the way it might be implemented (i.e. it is defined in terms of what it does, not how it does it). The Service Domain's discrete business function/responsibility is highly stable over time. As business practices and techniques evolve the inner working of a Service Domain may be enhanced with additional features. Extra service connections may also be needed but its foundational business role/purpose will remain unchanged.

   With good design a Service Domain can be built and integrated incrementally. Only the required functionality is developed as and when needed. This then can be continually enhanced/extended as the Service Domain is reused in other business contexts. As a result, Service Domain aligned solutions can enjoy a much longer shelf-life than conventional process based systems.

3. **Canonical or the 'same for everyone', supporting high levels of interoperability** – BIAN service domains define the generic functional building blocks that make up any bank. They can be compared to Solution Building Blocks as defined in TOGAF. Their role/purpose can be consistently interpreted from one deployment to another. They define business functionality that can usually be assigned to a responsible party in the organization that handle their operation and evolution/development. In different deployment situations a Service Domain may include a small proportion of functionality that is: location specific; more advanced; or unique/differentiating. But the Service Domain's core functionality is generic. In addition, its service connections, i.e. its position in the overall business blueprint is also generic.

   This means a bank or solution provider can switch out the underlying system for a Service Domain to replace it with an alternative solution. Once its service connections have been re-established (that will usually require some amount of detailed mapping and wrapping work) all other surrounding business activities should be largely unaffected.

In summary the value of the BIAN standard is to provide a conceptual business component framework for solution design and development. Its specifications define the standard functional building blocks in sufficient detail that they can be consistently interpreted in terms of the role of the Service Domain components and the intention

behind the service exchanges between them. The business information governed and exchanged by the Service Domains also define a high level business information architecture.

The high level conceptual BIAN specifications can jump-start development but the detailed (site-specific) implementation level designs are still required. Once built however, the Service Domain aligned systems will support incremental development, substantial operational reuse and where needed enable highly distributed and collaborative configurations. Furthermore, the migration to a service-based component architecture will progressively eliminate the excessive fragmentation and redundancy present in most banks' legacy application portfolios – an issue that today adds significant complexity and operational overhead to most banks' operations.

2.2 BIAN uses an 'Asset Leverage' Model View

The intent of a 'model view' of anything is to represent it in a concise format that highlights some specific properties or features of interest. For example, the map of a city highlights the roads and pathways for those wishing to get about, eliminating or greatly simplifying the representation of other aspects that could be distracting such as the precise structure of the buildings. Compare the A to Z© map of central London to a satellite image of the same location:

A map of a city is a view designed for navigation, highlighting roads and pathways

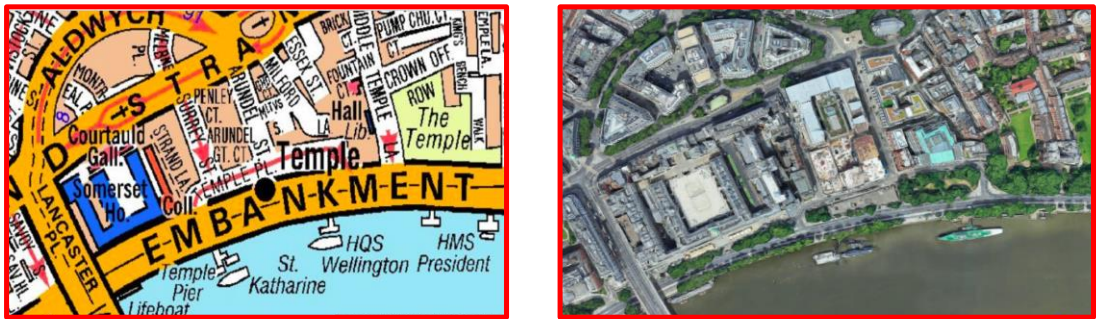


Figure 2 - Example Navigational Model View of a Town

It is important to be aware that the BIAN specification adopts a completely different model view of business activity than the more conventional process centric approaches that are widely applied today. This difference has far reaching implications for architects interpreting the BIAN standard.

Why has BIAN selected a different model perspective?

The conventional process oriented models represent business activity as a linked sequence of work tasks. This perspective is most useful for systems that automate these actions as a repeating workflow like a factory production line. A good proportion of banking activity (transaction processing in particular) suits this model view well where automation and straight through processing (STP) are the goals. But systems that support more interactive/collaborative workflows and that integrate advanced analytics to drive decision making can benefit from using a different model. For these types of systems, a component business model is far more representative.

The challenge for BIAN has been to select and apply a technique for defining a banking component model specifically suited to service oriented design: one that helps isolate a comprehensive collection of discrete (non-overlapping) business functional components and that also defines the service exchanges that occur between these operational 'building blocks'. Furthermore, as already noted, in order to be an industry standard the scope or role of each conceptual building block and its associated service exchanges needs to be 'canonical' in nature, i.e. consistently interpretable by all industry participants.

So what is the component model that BIAN uses? - an "Asset Leverage" model

The technique BIAN uses to isolate its banking components is referred to as an 'asset leverage' model. It is an empirical rather than theoretical technique meaning that a candidate component is first defined and then tested out in practice, modeling its behavior using real world scenarios to confirm and refine its definition. BIAN members have spent several years considering numerous business events in order to define a comprehensive collection of banking components, the "Service Domains", that today cover most banking activities.

The asset leverage model considers two aspects of an enterprise. The first aspect reflects that the business possesses/has access to things of value or 'assets'. These can be tangible things like buildings and computers, or intangible things such as relationships, knowledge/knowhow and goodwill.

The second aspect is that in day-to-day execution these assets are controlled or manipulated in specific ways (using IT control systems) to enhance and/or exact commercial value from them. The types of control applied can be categorized using different general "functional patterns", for example a computer is *operated*, a customer relationship is *managed*, market knowledge is *analyzed* to extract insights.

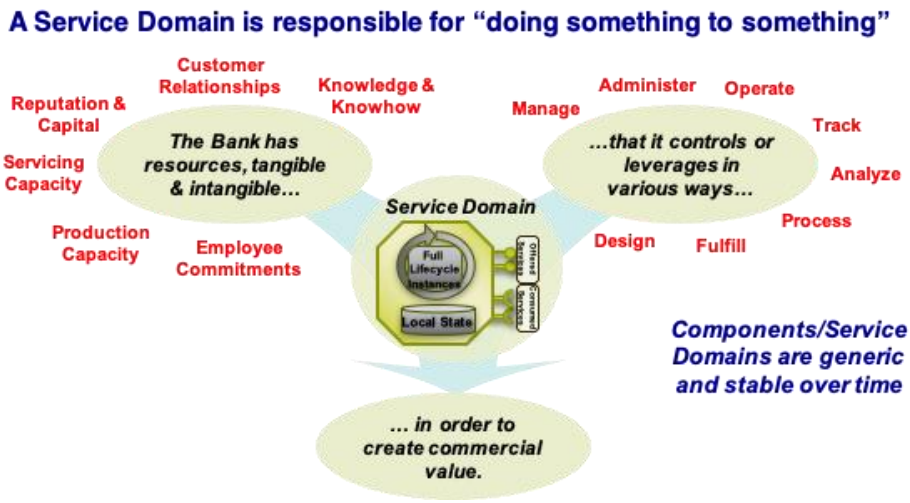


Figure 3 - A Service Domain Does Something to Something

A BIAN Service Domain is defined to be responsible for implementing one pattern of control to instances of just one type of asset. The Service Domain does this from start to finish for the complete life cycle, as often as required by the business.

Unlike the production-line process implementation where the steps in the processing logic are tightly linked together from end-to-end for one specific purpose, the BIAN Service Domain functional components can be implemented in a manner that allows them to be more loosely coupled together. They can be engaged in any suitable combination and sequence as necessary to address many different business events, in parallel if needed. These design properties are revisited in more detail with examples in the next section.

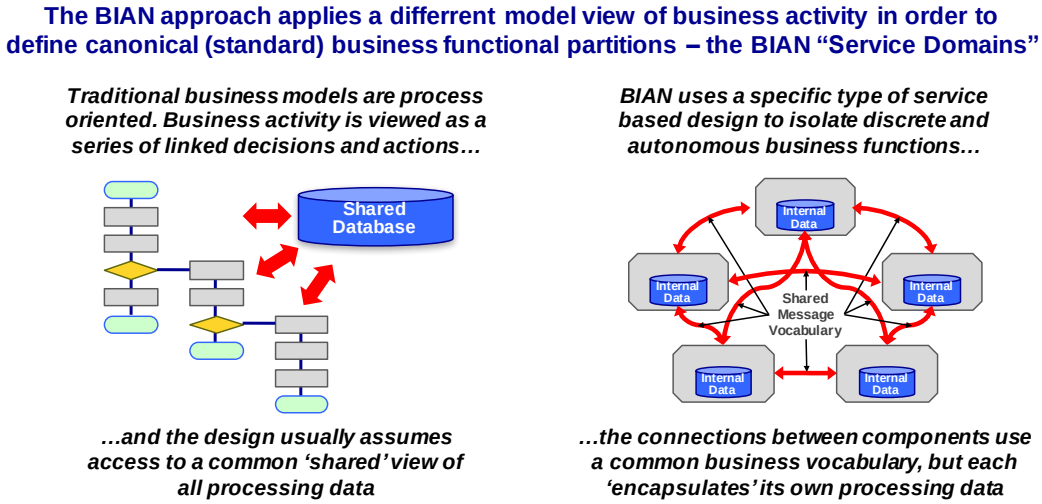


Figure 4 - Process Vs Component Model View

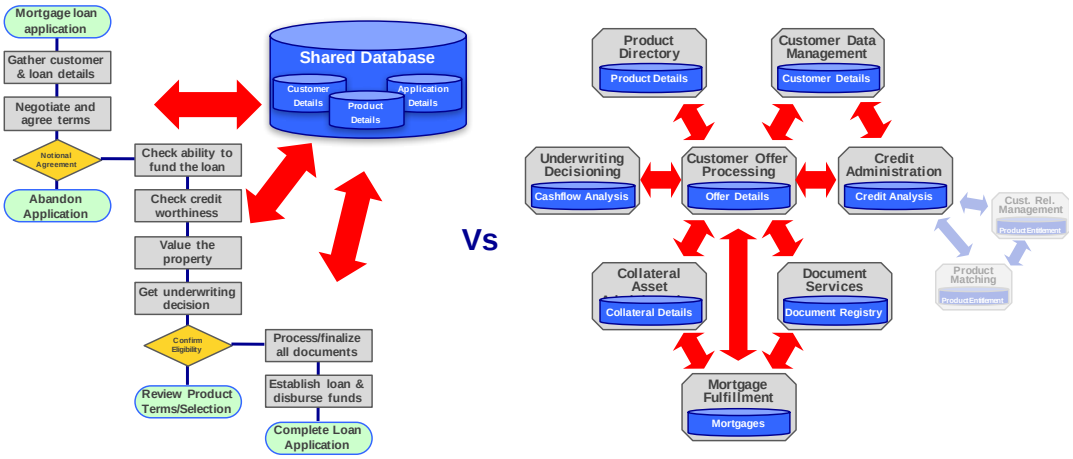
The Service Domains can represent concentrated centers of specific business expertise or capabilities that can be assigned. When appropriate they can be directly associated with responsible organizational units of the enterprise

A brief aside: *BIAN's original intent in selecting an approach was to help eliminate the complexity in the bank's application portfolio to improve interoperability within banks by defining a service oriented architecture (SOA). It is a convenient coincidence that the same BIAN Service Domain functional partitions define highly encapsulated and autonomous components that are well suited to implementation in the highly distributed cloud/container technical environments that are being increasingly adopted with the support of APIs today. Indeed, without a robust business partitioning approach such as that used by BIAN, it is extremely difficult to develop highly distributed systems of any scale mostly because handling shared business information quickly becomes excessively complicated.*

2.3 Component Vs Process Business Designs – an example for comparison

The difference between a more conventional process model view and the component type of model can be demonstrated using the simple example shown below for processing a mortgage offer made to a customer. In the process model on the left the workflow is broken down into ever finer grained actions that can then be programmed as an (partially) automated workflow. The component view on the right however simply defines a linked collection of specialized business functions (components).

An example shows the different model views for a mortgage application. The bank and customer first agree general terms and formalise details when a property is found…



The are many possible variations in the process view, but the components are the same

Figure 5 - Process Vs Component Model of a Mortgage Application

In the conventional process model the end-to-end linked sequence of actions is streamlined and where possible automated. Once the system is built to execute this defined series of automated tasks however, it may not be too easy to amend this sequence to handle different business situations that might subsequently evolve. In the example the mortgage is pre-approved and various checks are made before the property is found. What if later there was a new requirement to shift the approval decision until after the property is identified for some reason?

Conversely in the service-centered design represented using BIAN Service Domains all the involved specialized business elements are identified, but no specific sequence is inferred – they simply interact as and when necessary through triggered or orchestrated service exchanges. If they are correctly implemented the 'service centers' should be able to support many different processing sequences/variations with little or no change to the workings of the individual components.

In addition to the flexibility to support any suitable sequence or pattern of collaboration, the example highlights the potential for re-use of the operational capabilities. In the process model the logic and data is all embedded in the processing engineered to handle this one specific business event and it may not be easily separated out for re-use elsewhere. By contrast, in the service-centered model, each Service Domain can be designed to operate autonomously and be engaged in many different business events.

For example, the Credit Administration component that is responsible for determining the credit worthiness of the customer could be reused in many other business contexts such as product selection/matching during customer relationship development as indicated with the partly obscured components towards the right hand side of the diagram above.

A component model does not mandate a service-oriented approach…

It is important to make the distinction between the related concepts of *componentization* and *service enablement*. Componentization defines discrete business capabilities. Its value is to identify specialized business functional partitions that can be defined and refined in isolation and used and re-used in many combinations to support different business behaviors. Componentization is useful to eliminate operational redundancy, to streamline business operations and to support highly distributed/decoupled operating models and their supporting systems when appropriate.

Service enablement is an implementation option that can be applied to the (systems underlying the) components to support a more flexible/dynamic operating model, but sometimes introducing significant overheads in terms of service latency/performance and orchestration. Service enablement greatly suits applications in the front office where interactive collaboration is needed and the flexibility to mix and match capabilities can be leveraged. Conversely, for back office transaction processing where the activity is more repetitive and fixed-sequence in nature the component connections tend to be more permanent. Here throughput/performance is usually more important, off-setting the benefits of the flexibility provided by service enablement.

For the transactional systems the component view is still useful however, to better partition and decouple activities, to streamline, optimize and/or batch together processing. Standard ‘service aligned’ interfaces can also be used to import/export information. But transaction processing related exchanges within and between the transaction systems are typically better implemented as ‘hard-wired’ point to point interfaces rather than service enabled connections. More is said about this later…

2.4 The BIAN Service Domain – Some Example Definitions

BIAN’s membership has defined a collection of 320+ Service Domains over several years using real-world banking examples. The designs are based on a mutually exclusive, collectively exhaustive (“MECE”) hierarchical classification of some 250+ generic banking asset types combined with nineteen distinct patterns of commercialization behavior (called ‘Functional Patterns’ that include the three example behaviors: *operate*, *manage* and *analyze* briefly mentioned earlier). These design features are fully defined in the next section of this guide. Here some examples are used to describe the core architectural properties of the Service Domains in more general terms first.

As outlined earlier, a “Service Domain” represents the combination of one pattern of commercialization behavior being applied to instances of one type of asset. The Service Domain is also defined to be responsible for fulfilling this business function for the complete lifecycle and for as many times as might be necessary. Using the same three functional patterns again, the behavior of some example Service Domains is as follows:

> ***Systems Operations*** – is a Service Domain that *operates* a computer facility (such as a production application platform) from the time it is turned on to the time it is switch off – for as many operating sessions as might be required over the production life of the technology
>
> ***Customer Relationship Management*** - is responsible for the set-up, maintenance and execution of a customer relationship plan from the time the relationship is first established through to its termination. It does so for every active customer relationship that it *manages*
>
> ***Market Analysis*** – is responsible for consolidating the market information/research and applying whatever type of *analysis* might be required at any time to develop any particular market insight when requested

Though every Service Domain applies a single pattern of behavior to instances of a single type of asset for the complete lifecycle the above descriptions show that Service Domains may manifest a wide range of operational behaviors depending on their particular business role.

For example, some Service Domains may act on only one or a very few concurrent asset instances whereas others may be handling many millions of instances at the same time (compare Systems Operations that handles a single production system operating session at a time to Customer Relationship Management where there may be many millions of active relationships). Similarly, the life cycle for some may be extremely long where as others may ‘churn’ quickly (for example the Service Domain “Product Design” handles product design specifications that can persist for many years Vs. the Service Domain “Customer Contact” that handles call center conversations with customers that each may often last just a few seconds).

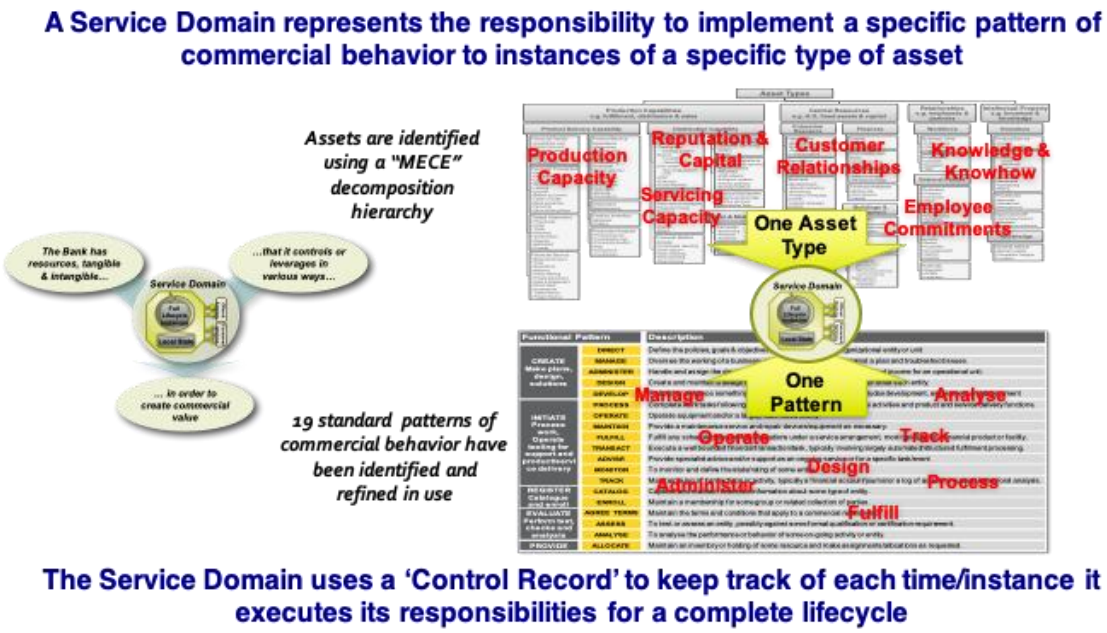


Figure 6 - The Functional Patterns and Asset Decomposition Hierarchy

The top level of the asset decomposition and nineteen general functional patterns that BIAN has used to develop Service Domains are shown schematically above. Note that it is not necessary for implementers to be overly familiar with these terms. But when considering the role/purpose of a Service Domain it can help to focus on its functionality by considering the main behavior it implements (its associated functional pattern) and the subject it acts upon (the asset type) as represented by the Service Domain's '*control record*'.

The Service Domain's Control Record

Every BIAN Service Domain specification defines a single associated operational artifact called its "control record". This is simply the mechanism it uses to control or trace the execution of one occurrence of it performing its business role for a complete lifecycle. The control record is an important feature of a Service Domain for development because it contains most of the key information that is likely to be referenced and exchanged in service operations between the Service Domains. As already described, the control record reflects the combination of the type of asset acted upon and the functional pattern being applied. Expanding on the three Service Domain examples already mentioned:

> ***Systems Operations*** – its control record is the system operating session that defines the schedule of actions taken and captures any details of operational

events that occur or tasks that are processed throughout a complete operating session cycle for the production system

Customer Relationship Management – its control record is the customer relationship plan/charter that sets out the goals, relationship development actions and records key events and performance over the complete duration of an individual customer relationship – often likely to be many years

Market Analysis – its control record is the market analysis perspective developed, including the detail of the information referenced and the analysis algorithms applied in order to develop specific insights for one analysis request

Other than general operating control and reporting information that every Service Domain maintains when implemented as a service center, the significant majority of the information accessed by other Service Domains through service exchanges is extracted from one (or more) of its active control record instances.

As noted, the Service Domains each support a discrete and non-overlapping business functional partition that collectively cover all banking activities. It follows that the discrete partitions of business information they each govern and exchange through service interactions (as defined by their control records) together defines a type of high level business information architecture.

Service Domain Service Operation Action Terms

The last key design feature of the Service Domain that is outlined in this section relates to the Service Domain's service interface. BIAN has defined a standard set of "action terms" that characterize the type of service operation exchanges that a Service Domain supports. The complete collection of action terms is also described later but using the three Service Domain examples one last time for a quick preview of some BIAN actions terms: (the standard action terms described here are *initiate*, *retrieve* & *evaluate*)

Systems Operations – its control record is the system operating session and an example service operation that might reference it would be a call to *"initiate"* an operational service or feature as would be defined and handled using information held in the control record. For example, to initiate an ATM withdrawal transaction from a teller device that is being operated on the ATM network

Customer Relationship Management – its control record is the customer relationship plan and an example service operation would be a call to *"retrieve"* an assessment of performance to plan for a particular customer relationship that would be maintained in their associated control record instance

Market Analysis – its control record is the market analysis perspective and an example service operation would be a call to *"evaluate"* some specific aspect of the market. The service call would result in the creation of a new control record instance that would contain the input market reference information, applied analysis algorithms and the eventual findings of the analysis that are finally returned

2.5 Service Domain Encapsulation

Because the Service Domain handles all activities for the complete life cycle it internalizes or encapsulates away much of the more complex processing logic and associated business information. Any other Service Domain calling on its services only needs to understand the externally visible information contained in any of its offered services. This typically involves a more easily interpreted sub-set of the complete set of information that a Service Domain uses.

For example, the Customer Relationship Management Service Domain as already described contains detailed analysis of product and service utilization and performance, a potentially lengthy schedule and detailed record of meetings, product utilization projections, relationship development ideas etc. But only an extract or higher-level interpretation of this information will be provided through its offered external services. A caller requesting details of a customer's profitability has no need to understand the many detailed aspects of relationship management to be able to interpret a simple relationship profitability performance report that is derived from all of this information.

The encapsulation of processing logic and information results in two different perspectives of the Service Domain's business information. There is the detailed and potentially extensive Service Domain specific logic and data needed to support its internal processing. Then there is also the shared/external business information that makes up the content of the services it offers to calling Service Domains – this represents a common business vocabulary that is typically a much more limited subset of the managed business information.

The detailed internal logic and data can adopt any suitable format/schema as it is not shared outside of the Service Domain and would typically be maintained on an internal/local database. The shared business vocabulary however needs to be defined consistently amongst all that access it.

The BIAN Business Object Model (BOM) that is under continual development is a conceptual data model that captures this shared business vocabulary. This provides a consistent definition of the control record content and the exchanged business information passed by services between the Service Domains.

Information Precision

The shared business vocabulary by definition covers concepts that span the business that will typically include the more general and widely recognized banking terms. This shared business vocabulary needs to conform to a common definition/specification. The required precision of this definition however varies depending on the nature of the information and the way it is handled.

Some shared business information requires a very precise definition – financial transactions for example are made up of elements/attributes with exacting specifications, such as the amount, currency, processing dates and involved accounts/counterparties. Other information has less inherent precision and can be represented in unstructured or variable formats (such as a customer's credit evaluation, product preferences, or a financial market performance analysis).

Furthermore, the required precision for the service exchange is very different if the end consumer is a machine versus a cognitive/human reader. A basic machine will only correctly interpret precise machine-readable data elements that have been fully specified in advance. An intelligent reader on the other hand is able to handle complex/variable formats and interpret and extract relevant information from unstructured presentations.

Traditionally the term API has referred to structured machine-to-machine connections. Service APIs are increasingly being defined to support screen-based dialogues where the provider or consumer or both is cognitive/human. This distinction is important when relating the BIAN service operations to underlying systems APIs as will be explained with examples later in this guide.

Section Summary

The objective of this section is to clarify that the BIAN representation of business as discrete functional components that can be service enabled is fairly novel and distinct from conventional process based designs. It explains that aligning development to the BIAN component boundaries can have significant beneficial architectural implications but recognizes that aligning to the specifications requires an investment of effort by developers in interpreting the BIAN specifications.

The section described some of the main properties of the foundational building block of the BIAN model – the Service Domain. It explained that each Service Domain's role is defined to be the application of a pattern of behavior (functional patterns) to instances of a type of business asset. It also outlined that BIAN defines standard types of service operations (action terms) offered by a Service Domain to others requiring access to its capabilities.

In the next section the different BIAN design artifacts are described in far greater detail.

3 – The BIAN Design Artifacts

This section describes the available BIAN design artifacts. These include:

1. The BIAN Service Landscape
2. BIAN Service Domain
    a) Functional Patterns
    b) Asset Types & Right-sizing Service Domains
    c) Control Records
    d) Behavior Qualifiers
    e) Service Operations and Action Terms
    f) Service Domain First Order Connections
    g) Service Domain Information Profile
    h) The Figure “8” Diagram
3. Business Scenarios (examples)
4. Wireframe Diagrams (examples)
5. BIAN Semantic APIs
6. Service Domain Event Triggering (Proposed Extension)

The BIAN standard combines the formal canonical designs of the Service Domains and their associated service operations with a wide collection of usage examples that help clarify the intended working of the Service Domains. Technically speaking the usage examples (presented in the form of ‘business scenarios’ and ‘wireframes’) are not part of the formal BIAN standard – they are not intended to be prescriptive but instead simply provide archetypal illustrations of how the Service Domains can interact. They are useful to help architects model their business requirements using the Service Domains.

The available usage examples (scenarios and wireframes) also provide technical leads and architects with a starting point for a design that can be adapted and extended to define more detailed business requirements for systems development. Because they are only examples, architects can change the available BIAN scenarios and wireframes to better match their own needs as long as when doing so they do not amend the underlying role/purpose of any involved Service Domains.

All published BIAN artifacts can be found on the BIAN public site (BIAN.org).

3.1 The BIAN Service Landscape

The ~320 Service Domains that have been identified so far by the BIAN membership are cataloged in a reference framework call the BIAN Service Landscape. The landscape organizes Service Domains into groups based on large *business areas* and within these areas more narrowly defined *business domains*. The layout is simply intended to help

with the identification/selection of Service Domains. BIAN has also created a second layout for this reference framework referred to as the “value chain” landscape format.

The original “matrix” landscape format organizes the Service Domains into groups with business areas (columns) and business domains (blocks) based on predominantly technical properties of the Service Domains. The value chain format defines a different classification of business areas and business domains that depict a more enterprise organizational view. The Value Chain layout is designed to be more intuitive for use by business practitioners.

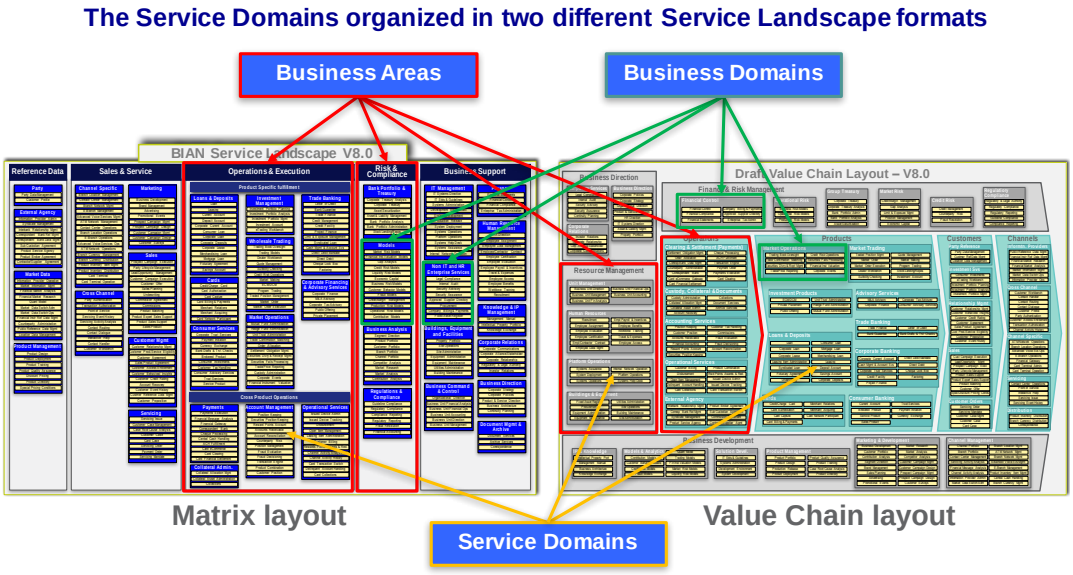


Figure 7 - Two Service Landscape Formats with Business Areas/Domains Highlighted

It is important to note that both landscape layouts contain one of each of the exact same collection of Service Domain components. With their dissimilar business areas and business domain definitions the two landscape formats group/arrange these Service Domains differently. Experience within the BIAN membership is that the value chain layout is generally preferred by business practitioners. It can be more readily related to a typical enterprise operating model or organization.

3.2 BIAN Service Domain Specifications

The building block of the BIAN model is of course the BIAN Service Domain. The main challenge for developers when using BIAN is to frame their business requirements in a form that uses collections of Service Domains exchanging services rather than a more conventional end-to-end process oriented design. The BIAN Business Scenario and BIAN wireframes are helpful design artifacts for making this transition. But before describing business scenarios and wireframes we set out the key design aspects of the BIAN Service Domain itself.

3.2.1 Service Domain Functional Patterns

A Service Domain is a conceptual functional design that can be mapped/related to a major application module. As already described a Service Domain's core purpose is that it controls the application of some type of commercialization behavior to instances of a type of asset. It does this from start to finish for as many occasions as called for by the business. The BIAN approach currently defines 19 general commercialization behaviors – called "functional patterns". Every BIAN Service Domain applies one of these functional patterns to instances of its assigned asset type. The list of BIAN Functional Patterns is shown in the table:

The BIAN Functional Patterns

Functional Pattern		Description
CREATE Make plans, design, solutions	DIRECT	Define the policies, goals & objectives and strategies for an organizational entity or unit
	MANAGE	Oversee the working of a business unit, assign work, manage against a plan and troubleshoot issues.
	ADMINISTER	Handle and assign the day to day activities, capture time worked, costs and income for an operational unit.
	DESIGN	Create and maintain a design for a procedure, product/service model or other such entity.
	DEVELOP	To build or enhance something, typically an IT production system. Includes development, assessment and deployment
INITIATE Process work, Operate tooling for support and product/service delivery	PROCESS	Complete work tasks following a procedure in support of general office activities and product and service delivery functions.
	OPERATE	Operate equipment and/or a largely automated facility.
	MAINTAIN	Provide a maintenance service and repair devices/equipment as necessary.
	FULFILL	Fulfill any scheduled and ad-hoc obligations under a service arrangement, most typically for a financial product or facility.
	TRANSACT	Execute a well bounded financial transaction/task, typically involving largely automated/structured fulfillment processing.
	ADVISE	Provide specialist advice and/or support as an ongoing service or for a specific task/event
	MONITOR	To monitor and define the state/rating of some entity.
	TRACK	Maintain a log of transactions or activity, typically a financial account/journal or a log of activity to support behavioral analysis.
REGISTER Catalogue and enroll	CATALOG	Capture and maintain reference information about some type of entity.
	ENROLL	Maintain a membership for some group or related collection of parties.
EVALUATE Perform test, checks and analysis	AGREE TERMS	Maintain the terms and conditions that apply to a commercial relationship.
	ASSESS	To test or assess an entity, possibly against some formal qualification or certification requirement.
	ANALYSE	To analyse the performance or behavior of some on-going activity or entity.
PROVIDE	ALLOCATE	Maintain an inventory or holding of some resource and make assignments/allocations as requested.

Figure 8 - BIAN Functional Patterns with Descriptions

For developers the functional pattern provides a clear indication of the core functionality provided by the Service Domain. To make functional patterns more easily interpreted a 'generic artifact' is associated with each functional pattern. This simply translates the action of executing the behavior into something more concrete (basically converting the behavior from verb to noun form). The generic artifact describes the type of artifact/document that is used/produced when tracking the actions of the service domain as it completes its execution from start to finish. The generic artifacts associated with each functional pattern are listed in the table:

The BIAN Functional Patterns and their Generic Artifacts

Functional Pattern	Generic Artifact
DIRECT	Strategy
MANAGE	Management Plan
ADMINISTER	Administrative Plan
DESIGN	Specification
DEVELOP	Development
PROCESS	Procedure
OPERATE	Operating Session
MAINTAIN	Maintenance Arrangement
FULFILL	Arrangement
TRANSACT	Transaction
ADVISE	Advice
MONITOR	State
TRACK	Log Record
CATALOG	Directory Entry
ENROLL	Membership
AGREE TERMS	Agreement
ASSESS	Assessment
ANALYSE	Analysis
ALLOCATE	Allocation

Figure 9 - BIAN Functional Pattern Generic Artifacts

3.2.2 Service Domain Asset Types & Right-sizing Service Domains

The other key facet that defines the functional scope of the Service Domain is the asset type that it acts upon. An asset in this context is something of inherent value/purpose that the bank owns or at least has some influence over. Assets can be tangible things like computers and buildings or they can be far less tangible things such as relationships, knowledge and knowhow.

The asset classification used by BIAN breaks down to some 250-300 discrete asset types. Note that in BIAN's asset type classification the ***capacity to perform*** a business action – such as *service customers' needs* or *fulfilling banking products and services* is treated as an asset type. The top level categorization of asset types used in the BIAN specification is as follows:

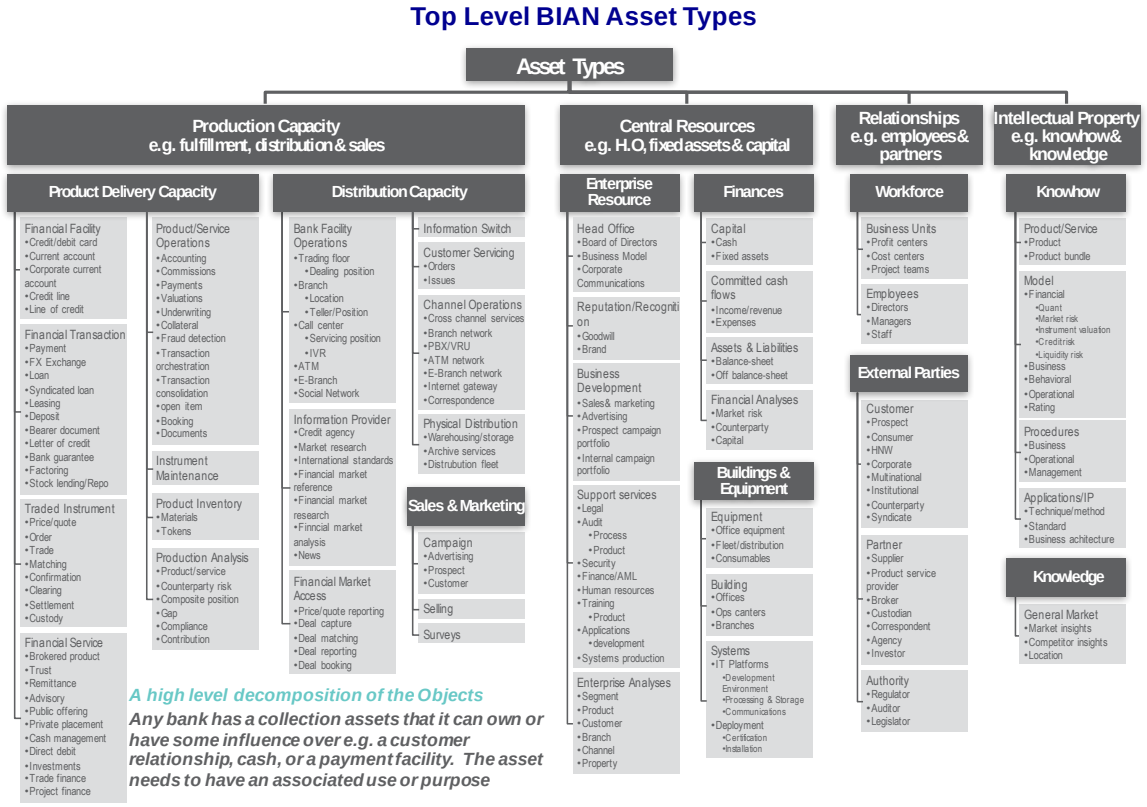


Figure 10 - Top Level BIAN Asset Types

Right Sizing Service Domains

Right sizing is key to ensure the defined Service Domains are 'elemental' – meaning that a bank either needs the Service Domain in its entirety or it does not, i.e. it can't split the Service Domain to adopt only some subset of its core function. Defining elemental Service Domains is necessary for the BIAN standard to be canonical: i.e. designs that are consistently interpreted in different deployments. (If it is possible to apply just part of a component its specification quickly ceases to be standard…)

The right sizing technique used to define Service Domains is quite complex and is not something that technical leads and architects need to learn to apply themselves as all the Service Domains they need should already be fully specified. The technique is properly described in BIAN architectural guidelines and training materials. An outline description of the technique is included for reference as Attachment B to this guide for convenience.

Briefly the right sizing technique involves decomposing the types of assets to the lowest granularity for which they retain unique business context or ownership. Below this level of granularity functions that act on the assets become more utility in nature.

For example, a classification decomposition of the banks production systems that are operated can be made to the level where individual systems are identified such as the ATM network, contact center support application or the internal office network. Further decomposition of these systems would start to identify their constituent processing functional features. In many cases these features are not uniquely assignable to a single responsible party in the enterprise's organization.

Each of the example systems listed above is likely to include some function to add/register some new entity (which would be a new ATM device, a customer servicing position and an employee reference respectively for the listed examples). This 'register' function is more of a utility feature common to all or at least most systems. Operating a generic 'entity registration function' would not really be a uniquely assignable business responsibility in the way that operating an ATM network is.

Comparing Operational Vs Utility re-use

The two different types of re-use are both important but apply in very different contexts. The assignable role performed by a Service Domain – for example Customer Relationship Management and Document Services (that handles the classification, storage and distribution of important documents) represents a re-usable operational capability. Operational re-use involves parts of the business using shared services provided by other specialized business units to gain access to their capabilities.

The finer grained functions making up Service Domains that may include recurring elements define reusable *utilities*. For example, a frequently performed product pricing calculation can be implemented as standard software utility. This utility can then be built into multiple product processing systems. It makes great sense to define and implement standard solutions for commonly executed functions for consistency and to avoid constantly re-writing code. But this is of course a different type of re-use:

- Service Domains define discrete operational business capabilities that can be assigned to a responsible party in the organization and reused by other parts of the business as they undertake different business activities

- Utility functions define standard actions/behaviors that can be encoded in re-useable code modules (using SW procedure libraries, micro-services etc.). The deployed instances of utility functions execute completely independently of each other

The important distinction between Service Domain operational capability reuse and software utility reuse is revisited in more detail in later sections of this guide.

3.2.3 Service Domain Control Records

The combination of the generic artifact and asset type defines a Service Domain's "control record". The control record specification is of particular interest as it comprises the main business information governed by the Service Domain. It can contain a very broad collection of information as it includes all the information needed to control processing, any information that might be referenced and also any information that is generated by the Service Domain as it completes a full cycle of its work.

An indication of the type of information that might be found in a control record is shown with an example control record for the Party Authentication Service Domain that handles confirming the identity of a customer.

The BIAN Control Record for the Party Authentication Service Domain

Attribute Cate	Level 1	Level 2	Level 3	Level 4	Level 5	Description
CR Party Authe	Party Authentication Assessment Instance Record					The authentication assessment combines the results of one or more tests to determine the level and authentication grant as
		Customer Reference				Reference to the customer as the authentication subject
		Party Reference				Reference to the party or legal entity as the authentication subject
		Party Authentication Assessment Profile				Details the types of authentication assessments that are combined into the overall evaluation
			Authentication Type			Reference to the different types of authentication assessment
		Party Authentication Consolidation Record				The combination of the different assessment results used in the authentication determination
		Customer Contact Authentication Level				The required value and value returned as a result of the authentication task, defining the level of identity assurance achieved -
BQ Password Instance Reco		Password Instance Record				Authentication using reference data and submitted passwords that are checked against records maintained by Issued Device
			Authentication Reference Data Reference			Reference to a customer reference data item to be compared with submitted value
			Authentication Reference Data Type			Defines reference data item type submitted for comparison
			Authentication Reference Data Value			The customer provided value is matched to the bank's maintained value
			Authentication Password Reference			Reference to an issued password
			Authentication Password Template			Defines allowed values/format for an issued password
			Authentication Password Stored Value			The bank maintained value (can be customer provided or a bank generated value - encryption applies)
			Authentication Password Valid From/To Date			The valid period for the stored password
			Authentication Password Presented Value			The customer provided value is matched to the bank's maintained value
			Authentication Password Test Result			The result of the reference data and password checks
BQ Question Instance Recor		Question Instance Record				Authentication using secret questions that are checked against maintained values
			Authentication Secret Question Reference			Reference to the selected secret question
			Authentication Secret Question Template			Template includes the question text and provided customer response - given value is compared to the stored value
			Authentication Secret Question Value			This is the stored value, the provided value is compared to this
			Secret Question Test Result			The result of the secret question check
BQ Document Instance Rec		Document Instance Record				Authentication by reference to documents - typically 'government issued' that are kept on file
			Authentication Document Reference			Reference to the document and document type being presented for authentication
				Authentication Document Content		The submitted document content in any appropriate format/media (e.g. scan)
			Document Directory Entry Instance Reference			The document reference for the authentication document
				Document Content		The stored document - available in any suitable media for comparison
			Authentication Document Comparison Test Result			The result of comparing the presented document to the stored value
BQ Device Instance Record		Device Instance Record				Authentication by device reference, covers all devices (e.g. card, key-fobs, key-pad)
			Authentication Device Reference			Reference to the device being used for authentication
			Authentication Device Property Value			Property of the device being used to authenticate (e.g. phone number, URL)
			Issued Device Instance Reference			Reference to the customer issued device
			Issued Device Property Value			The registered customer device properties - maintained by SD-Issued Device Administration
			Device Test Result			The result of the device check
BQ Biometric Instance Record						Authentication using biometric such as face recognition, signature
			Authentication Biometric Type			The type of biometric record being used for authentication
			Authentication Biometric Record			The biometric record submitted for authentication (e.g. face scan, fingerprint, signature)
			Registered Biometric Instance Reference			The registered customer biometric record reference - maintained as an issued device instance
			Registered Biometric Instance Record			The registered customer biometric record - maintained by SD-Issued Device Administration
			Biometric Test Result			The result of the biometric check
BQ Behavioral Instance Record						Authentication based on detected and matched activity/behavior
			Authentication Behavioral Type			The type of behavioral record being used for authentication
			Authentication Behavioral Record			The behavioral record submitted for authentication
			Registered Behavioral Instance Reference			Reference to the registered customer behavioral record
			Registered Behavioral Instance Record			Registered customer behavioral record - maintained by SD-Issued Device Administration
			Behavioral Test Result			The result of the behavioral check

Figure 11 - Excel Extract of Service Domain Control Record

The BIAN standard provides initial control record information definitions for the Service Domains that can be filtered and expanded in the context of a specific implementation project. For those with object oriented design experience the control record can be considered as a type of ‘class’.

3.2.4 Control Record Behavior Qualifiers

Early experience using the BIAN Service Domain’s service operations to access its control record revealed that accessing the complete control record in a single service exchange did not always define a sufficiently narrow business context or purpose for the service operation to have an unambiguous definition (service operations are fully described later in this section). For example, a service operation to ‘execute’ some action against an active customer’s current account could have many different intended uses in different business contexts with different results (e.g. to execute a payment from the account or to execute a deposit into it). A further level of detail breakdown is

therefore required for the control record so that the accessing service operations can have a sufficiently singular/unique purpose.

In order to add this further level of detail BIAN uses *'behavior qualifier types'* to break down the work performed by the Service Domain as captured in its control record. A specific behavior qualifier type is defined for each functional pattern – the type defines how the pattern of behavior can be subdivided into its finer grained activities. Most important the behavior qualifier type retains the core behavioral characteristics of its associated functional pattern. In essence the overall work done by a functional pattern is made up of the collection of the same type of work done by its finer grained behavior qualifiers - the BIAN behavior qualifier design actually implements a fractal pattern.

The behavior qualifier types used to break down each of the BIAN function patterns are shown in the table:

The BIAN Functional Patterns, Generic Artifacts and Behavior Qualifier Types

Functional Pattern	Generic Artifact	Behavior Qualifier Type	Example
DIRECT	Strategy	Goals	Increase market share
MANAGE	Management Plan	Duties	Relationship development, Troubleshooting
ADMINISTER	Administrative Plan	Routines	Time-sheet recording
DESIGN	Specification	Aspects	Business requirements
DEVELOP	Development	Deliverables	Functional module specification
PROCESS	Procedure	Worksteps	Invoice generation
OPERATE	Operating Session	Functions	Message capture/routing
MAINTAIN	Maintenance Arrangement	Tasks	Preventive maintenance task
FULFILL	Arrangement	Features	Current account standing order
TRANSACT	Transaction	Tasks/Steps	FX pricing, market trade, clearing & settlement
ADVISE	Advice	Topics	Tax advice, Corporate finance
MONITOR	State	Measures	Composite position, Customer alert
TRACK	Log Record	Events	Customer life event, Servicing event
CATALOG	Directory Entry	Properties	Product pricing rules, Customer reference details
ENROLL	Membership	Clauses	Qualification/membership purpose
AGREE TERMS	Agreement	Terms & Conditions	Required disclosures,
ASSESS	Assessment	Tests	Password verification
ANALYSE	Analysis	Algorithms	Average balance calculation, Propensity to buy
ALLOCATE	Allocation	Criteria	Staff assignment, Facility allocation

Figure 12 - Functional Pattern/Generic Artifacts and Behavior Qualifier Types

Though a single general *behavior qualifier type* is associated with each functional pattern, the actual *behavior qualifiers* defined for a Service Domain will be particular/specific to the Service Domain. For example, a Service Domain with the functional pattern 'process' has the associated behavior qualifier type 'work steps'. The actual work steps defined for a 'process' Service Domain will reflect its own specific business role. The work steps that make up the processing for the Customer Billing Service Domain reflect how it processes a customer bill, i.e. : customer invoice

generation; invoice transmission/dispatch; payment tracking; and, payment processing work steps.

Currently BIAN only breaks down Service Domain control records to define the first level of behavior qualifiers as part of the standard definition (a position that may be revised with further deployment experience feedback). In the majority of cases this is sufficient to define the payload of a discrete service operation unambiguously. For some Service Domains with extensive information or functional content solution architects may find it necessary to define additional levels of 'sub-qualifiers' that break the control record down further to define suitably focused service operations. One example of possible sub-qualifiers is shown for the Party Reference Data Directory Service Domain with its functional pattern 'catalog', generic artifact 'directory entry' and behavior qualifier type 'properties'.

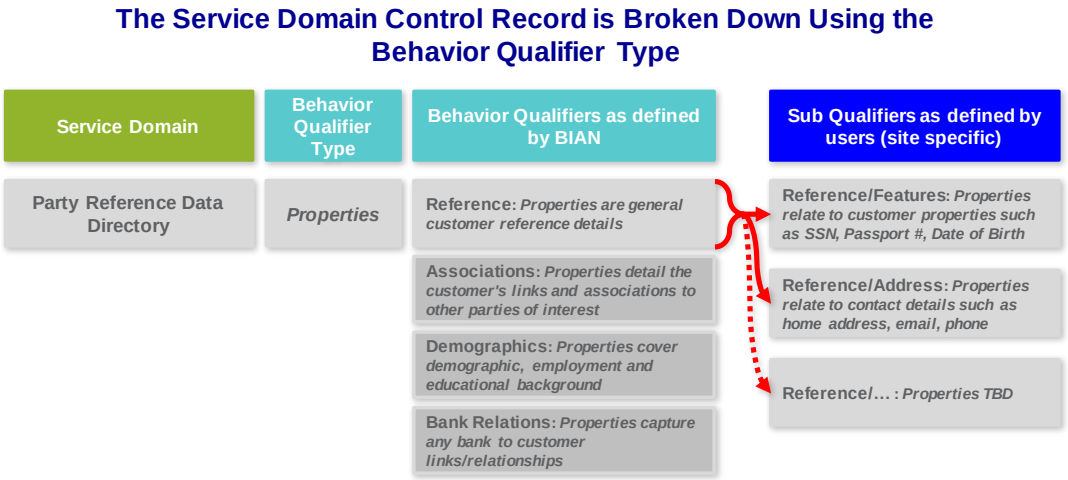


Figure 13 - Party Reference Data Directory Control Record

Note that it is important that as the control record is broken down that the applicable behavior qualifier type is applied consistently to define the sub-qualifier partitions and that the partitions are defined to be discrete and mutually exclusive & collectively exhaustive (MECE) at each level of decomposition.

3.2.5 Service Domain Service Operations & Action Terms

Every Service Domain offers a collection of service operations and usually consumes or 'delegates' by calling the service operations of other Service Domains as needed to complete its work. Here we consider the Service Domain's offered service operations.

BIAN has defined a general set of “actions terms” that characterize the purpose of an offered service. The collection of action terms is intended to be non-overlapping and collectively to cover all of the main types of service exchange that any Service Domain might support. The current set of BIAN action terms is defined as follows:

The BIAN Service Domain Action Terms

Action Terms			Description	Semantic API Interpretation
Service Domain (PUT,POST)	Actions to assign, set-up and configure a production capability	Activate	Assign/establish a production capability (applies to all Service Domains	Only called to establish steady state working (rarely used)
		Configure	Change the operating parameters for a service/capability – applies to defaults and active instances	The operational state may be re-configured typically to deal with processing exceptions and operational problems
		Feedback	Capture performance feedback. Note this can be at a Service Domain, Control Record or lower level	The term is used consistently to provide management/operational feedback (no production/transactional role)
Instantiation (POST)	Actions that trigger the generation of a new control record instance	Create	Create a new strategy, plan, approach, or develop a new design/model or solution	Several Service Domains are responsible for creating new instances
		Initiate	initiate a defined action: e.g. an operating session; procedure; set up a facility; or, a transaction	Most frequently used to start a production service/task, both transactional and back-office/support in nature
		Register	Classify and capture details of an entity in a reference directory	Several reference ‘directories’ are maintained – for example product, customer and employee related
		Evaluate	Perform an evaluation, including: a measurement; test/check; and, an analysis (can be on-going)	A range of different types of evaluation are supported by different patterns
		Provide	Assign or allocate resources or facilities from an inventory of available/tracked resources	Some resource pools are managed and assignments tracked (such as staff and building facilities)
Invocation (PUT,PATCH)	Actions that act upon an active control record instance(s)	Update	Change the value of some (control record) properties	Most control record instances are subject to update
		Control	Request for the processing to perform in a specific way e.g. block/suspend/skip/prioritise an action	Many Service Domain’s control record processing may be externally influenced e.g. wind-up, block/suspend, skip task
		Exchange	Provide input into the handling of an instance, typically in response to a question from processing	Many Service Domain’s control record processing may include key external exchanges e.g. to reject/approve/verify
		Capture	Capture transactional activity details against an instance: e.g. log an event/action, record usage	Some Service Domains support structured logs – for transaction tracking and/or events for later analysis
		Execute	Execute an automated/structured a task or action on an established facility/control record instance	Many automated tasks or actions may be initiated against active instances
		Request	Request a task that involves judgement/decisions/workflow against an established facility/instance	Many work requests may be applied to active instances (note these usually involve customer/workforce exchanges)
		Grant	Seek authority/grant to perform an action or use a resource that is overseen/governed by the SD	A small number of specialised Service Domains provide some form of permissions/authorization to a calling party
Reporting (GET)	Actions to extract/subscribe to recporting	Retrieve	Return information/a report as requested	Formatted and unformatted information can be requested in a wide range of situations (note there is no state change)
		Notify	Provide details against a predefined/subscribed to notification agreement	Service Domains may subscribe to information updates in a wide range of protocols/formats (for referential alignment)

Figure 14 - Action Terms with Definition and Examples

Those familiar with the BIAN standard will note that the action terms have undergone some minor revisions and additions since the Service Landscape Version 7.0 release. This is based on early feedback from members implementing the BIAN Semantic APIs. The changes have been required better to align the BIAN service operations to REST end point specifications as described in more detail later in this guide.

As can be seen in the table the BIAN action terms can be grouped in four main categories:

1. Those that act on/influence the operation of the Service Domain overall as a service center (*action terms: activate/configure/feedback*)
2. Those that result in the creation of a new control record instance, i.e. trigger a new life-cycle (*action terms: create/initiate/register/evaluate/provide*)
3. Those that act on an existing control record instance – typically invoking some function and/or changing/updating its state in some way (*action terms: update/control/exchange/capture/execute/request/grant*)
4. Those that obtain or subscribe to information updates for one or more control record instances. Importantly these actions do not change the state of the instance in any way (*action terms: retrieve/notify*) – Note: making this distinction is intended in part to help with Command Query Responsibility Segregation (CQRS) type deployments

BIAN defines the service operation to indicate a service dependency between the Service Domains – it does not presume any specific choreography/protocol for the exchange. So for example in implementation the service exchange could be a one-way flow of information or an instruction, perhaps with some simple acknowledgement of receipt. It could be a complex iterative dialogue as the request is refined based on interim exchanged details. Furthermore, the response could be immediate or there could be a significant delay requiring either or both the caller and provider to monitor for the response. The BIAN service operation also only details the exchange of information (which can include instructions and responses) but does not track the actual movement of physical items other than by implicit descriptions (such as the movement of physical currency or the deployment of resources).

Default Service Operations

When defining the service operations for the Service Domains BIAN has discovered in practice that there are sensible combinations of actions terms that apply for different functional patterns. For the 19 functional patterns and 17 action terms BIAN currently defines a default set of service operations. These are simply the defaults and it is possible that they do not all apply in some deployments or that there are practical situations where additional service operations are required that do not correspond to the defaults. The mapping is simply a starting point for architects to consider and is also used to define the service operations reflected in the BIAN Semantic API Portal.

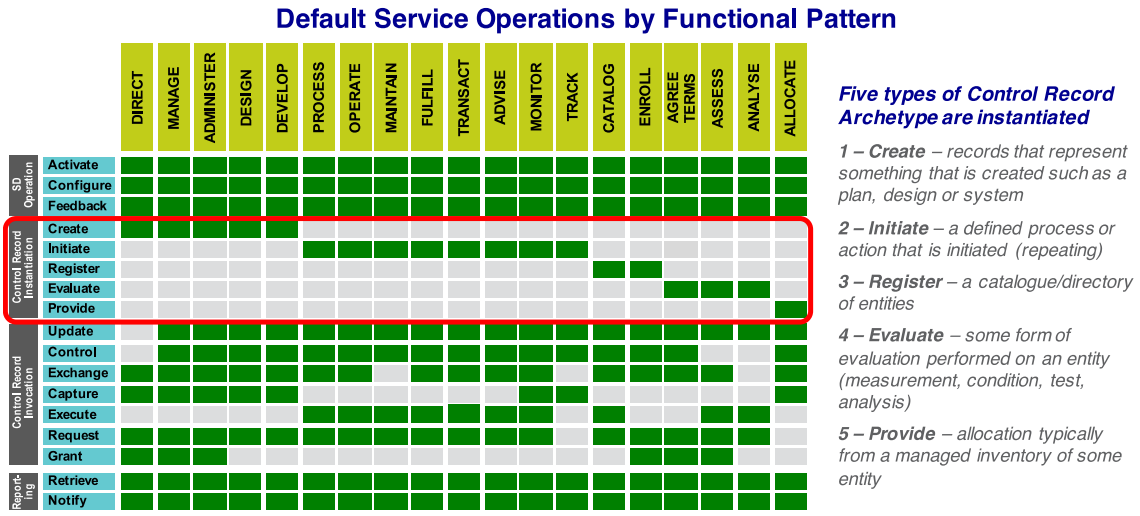


Figure 15 - Default Action Terms

A few interesting patterns can be seen in the default mappings that are worth a brief mention:

1. The action terms that control the overall Service Domain (*activate, configure* and *feedback*) apply not surprisingly to all Service Domains regardless of their functional pattern
2. Only one of the five action terms that result in the creation of a new control record instance applies to any one functional pattern. Again this is not surprising, but the nature of the control record instance archetype created is rather different for each of the five applicable action terms (as highlighted and described in the diagram)
3. Of the actions terms that act on an active control record (or a subordinate behavior qualifier) instance, most can apply for all functional patterns with just a few obvious exceptions
4. The *retrieve* and *notify* action terms also unsurprisingly apply in all cases regardless of the functional pattern. Note that the *retrieve* and *notify* action terms can be applied at the Service Domain level as well as the control record and behavior qualifier level as necessary to obtain different types of information extract/report

Sometimes it can seem that the same action term results in a fundamentally different response from different Service Domains. Though action terms are indeed consistent in their application, the apparent variation to the response is because Service Domains have very different underlying operational characteristics. An example that compares the

response to the *initiate* and *execute* action terms for a Service Domain with a *fulfillment* functional pattern and one with a *processing* functional pattern is presented as Attachment A to this guide to demonstrate this.

Service Operation Specifications

The actual service operation specifications for a Service Domain in the current release of the standard are defined using the applicable action term and optionally a behavior qualifier. Each service operation's payload is specified as an organized list of sematic attributes covering the key business information provided and returned for the service exchange. The precise format as applied for the BIAN Semantic APIs compliant to the REST architectural style is set out in more detail later in this section.

Note: in earlier releases of the BIAN Standard the Service Domain service operations were defined with four attribute types (Identifiers, Depiction, Instructors & Analysis). This structure has since been replaced with the more practical and comprehensive format used for the BIAN Semantic APIs.

3.2.6 Service Domain First Order Connections

The first order connections for a Service Domain capture any identified service connections required between it and both calling and called Service Domains. Each first order connection defines a service dependency between a single calling and called Service Domain that uses one available service operation (i.e. one action term and if appropriate one behavior qualifier).

A Service Domain's list of first order connections as captured in the BIAN standard will not necessarily ever be complete as the connections are discovered empirically by modelling business activity. It is likely that some viable business behaviors may not be fully anticipated. The known first order connections are useful for architects as they reveal the connections required to handle different business requirements and can help understand the overall scope/purpose of the Service Domain and provide the business context for its offered services.

First order connections can be associated with a (first order) business event. The event defines the external trigger that causes a call to the Service Domain's offered services. Many different Service Domains may call the same offered Service and each of these associations represents its own first order connection and associated business event. For example, many different Service Domains may request a customer's Current Account balance using the same 'retrieve' service operation offered by the Current Account Service Domain. But each call will be for their own specific purpose and each defines a first order connection.

In order to process a service request or in the course of its own internally scheduled activity the called Service Domain will also typically need to delegate actions, i.e. call on the services of other Service Domains. The first order connections capture both the offered and delegated service operation exchanges for the Service Domain. No formal link or association between an offered service and any dependency on delegated service calls is maintained however as this would contradict the foundational SOA principle of encapsulation.

Note: first order connections are used to assemble the BIAN business scenarios and wireframe views that are described later in this section. BIAN strives to capture all first order connections for Service Domains in the standard model as they are discovered through different requirement modelling efforts.

The practical exceptions where it is useful to model second order or 'nested' service exchanges are for Service Domains that need to perform their respective roles concurrently. This is the case for the Service Domains that handle customer interactions with the bank as shown in the example:

An Example of 'nested' service exchanges

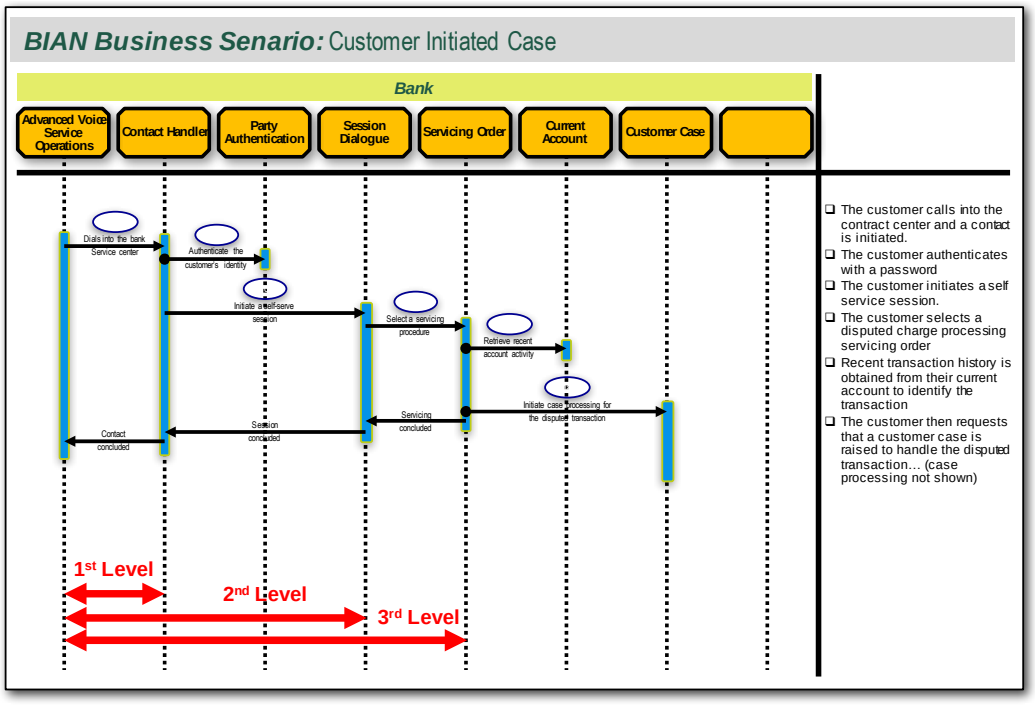


Figure 16 - A Business Scenario with Nested Service Exchanges

As can be seen in the business scenario, the Advanced Voice Services Service Operations Domain (that operates the PABX) calls the Contact Handler Service Domain that calls the Session Dialogue Service Domain, that itself calls Servicing Order Service Domain to process the customer's request. As a result of their integrated start/end dependencies there are three levels of nesting required in the scenario.

3.2.7 Service Domain Information Profile

The control record instances already described contain the primary information of interest for developers that is maintained by the Service Domain. Control record instances are typically accessed by the service operations for most transactional business activity. The Information Profile however describes the complete make-up of the business information governed by any Service Domain when implemented as a stand-alone service center. The information profile make-up is:

- information used in the control and management of the Service Domain as a service center including local copies of referenced information, accessed/delegated service details, resource administration, service domain activity and performance records and offered service definition and service configuration settings
- collective views and analyses of the collection/portfolio of control record instances including, usage, performance and historical analysis as might be required
- the content of individual control record instances, further broken down using behavior qualifiers as necessary.

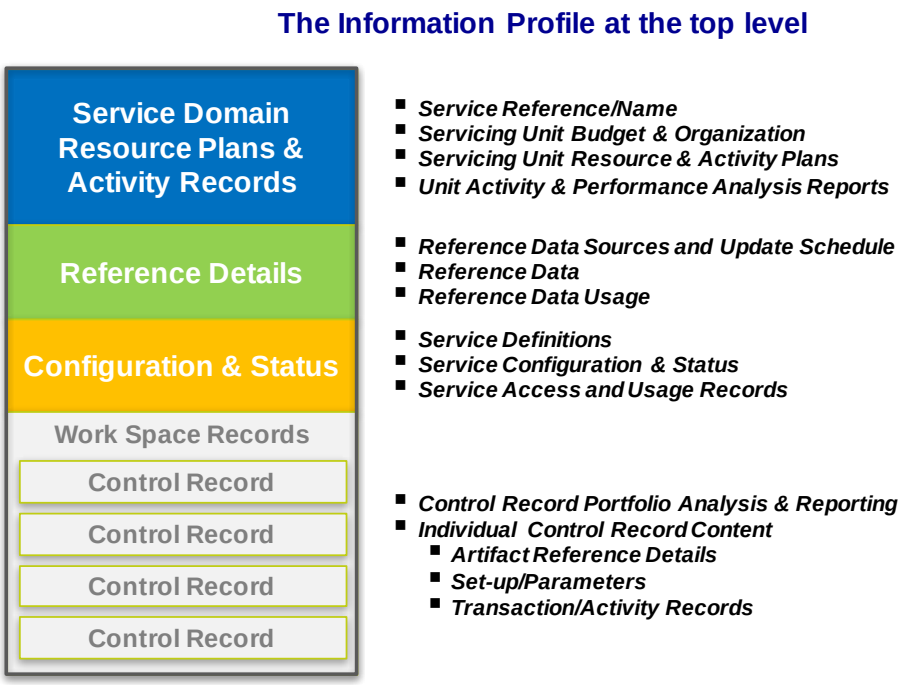


Figure 17 - The Information Profile – Top Level with Content Descriptions

As noted when the control record is broken down using the behavior qualifier type, the resulting partitions have the same characteristics as their parent partition – it in essence applies a fractal pattern. The property is particularly useful as it means the action term for a service operation is applied consistently to the control record, a behavior qualifier partition or any further sub-qualifier partitions. The use of the behavior qualifier type provides a mechanism for adding increasing precision in terms of defining the scope of the referenced information within the control record.

The Service Domain embeds business information in a Fractal Pattern

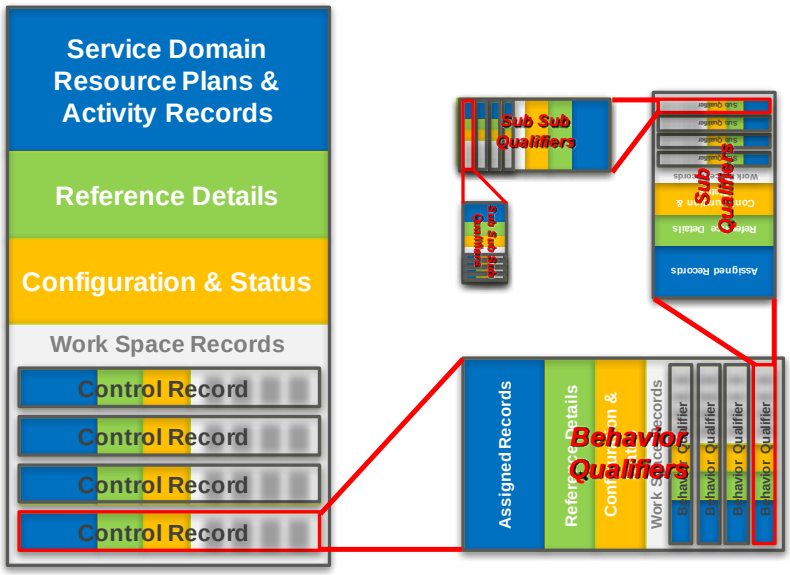


Figure 18 - The Fractal Nature of the Information Profile

At this stage BIAN has defined Service Domain specific descriptions of the make-up of the control records with their individual behavior qualifiers. This is the current focus for the BIAN membership because it defines the payload of the service operations most widely used in the API specifications for transactional banking activity. For completeness BIAN has also provided general descriptions/checklists for the type of information that can be anticipated covering the operational control and activity/performance analysis of the Service Domain that is used in more general management and control. All of the information definitions are semantic and subject to the following qualifications and limitations:

- **Only Covers Mainstream Behavior** – the information definitions relate to the prevailing mainstream functions performed by the Service Domain – they are intended to be indicative such that they support an unambiguous definition of the core role/purpose of the Service Domain and its service operation exchanges. They do not attempt to be exhaustive for example covering regional variations or more advanced/specialized distinctions. Furthermore, all activity considered is 'happy path' so error processing and exceptions are not generally considered

- **Only Provide High Level Semantic Definitions** – BIAN is a business architecture level conceptual specification and as such its information attributes are defined using fairly high-level semantic descriptions. In some cases, the level

of detail provided gets close to implementation level granularity. But in most areas the attributes defined by BIAN need to be related to more detailed information structures and definitions by the architect/developer. For example, BIAN might define an 'account statement' with properties such as 'period covered' and 'types of transaction included'. This is unambiguous in terms of the business requirement, but clearly far from a detailed specification of a physical statement report as might be printed off by an application

- **BIAN's Descriptive Definitions differ from Standard Data Formats** – The BIAN information attributes are defined in semantic/descriptive terms, BIAN does not provide a formal data definition. This is important to ensure that the BIAN specification is implementation agnostic – i.e. the BIAN information attribute can be mapped to any appropriate data representation. Consider for example the BIAN information attribute *'customer reference'*. This defines an attribute that provides some unique reference to a bank customer – an information concept that can be consistently interpreted. How that reference is subsequently realized in any specific implementation's data standard is not defined. An industry standard such as some variation of the IBAN code could be used or the developer might define their own unique bank specific customer key

- **Mapping to ISO20022 & Other Standards** – BIAN's policy is not to develop competing content with other prevailing industry standards. The current focus for BIAN is to map its semantic information attributes to the ISO20022 Business Model. Given that the scope of the ISO model is not complete in some areas covered BIAN has to define its own conceptual object model and map this to ISO20022. BIAN will also map to other existing standards as appropriate.

  At this time the BIAN specification comprises three related information descriptions:
  - the Service Domain Information Model comprises the semantic attributes that make up the Service Domains' information profile – primarily the control record definition;
  - the BIAN Business Vocabulary provides descriptions of the different information attributes (adopting industry accepted definitions where available); and
  - the BIAN Business Object Model maps the information attributes to the conceptual business objects for definitional consistency

As noted in an earlier section of this document (Section 2.5) the encapsulation property of Service Domains results in two overlapping views of business information – the first a higher-level conceptual information vocabulary that is passed as the payload of service

operations. Second the potentially far more detailed processing logic and data schema used in the internal working of the Service Domains. The focus for the BIAN standard is on addressing the first view: to define the business information that is shared through service operation exchanges between the Service Domains.

The goal of the BIAN Information Model specifications and the underlying Business Vocabulary and BIAN Business Object Model (BOM) is to provide a complete and consistent set of definitions at a level of detail where mappings to implementation level data standards and physical specifications can be done unambiguously and consistently. This is a major undertaking not least because of gaps and limitations in existing information/data standards across the industry.

BIAN will continually add detail and coverage to an appropriate level based on practical experience. In the interim architects and developers should expect that available semantic information definitions are only indicative and in many cases will be limited to high level/generic descriptions.

3.2.8 The Figure "8" Diagram

To conclude this section covering the description of Service Domain design, its main properties are:

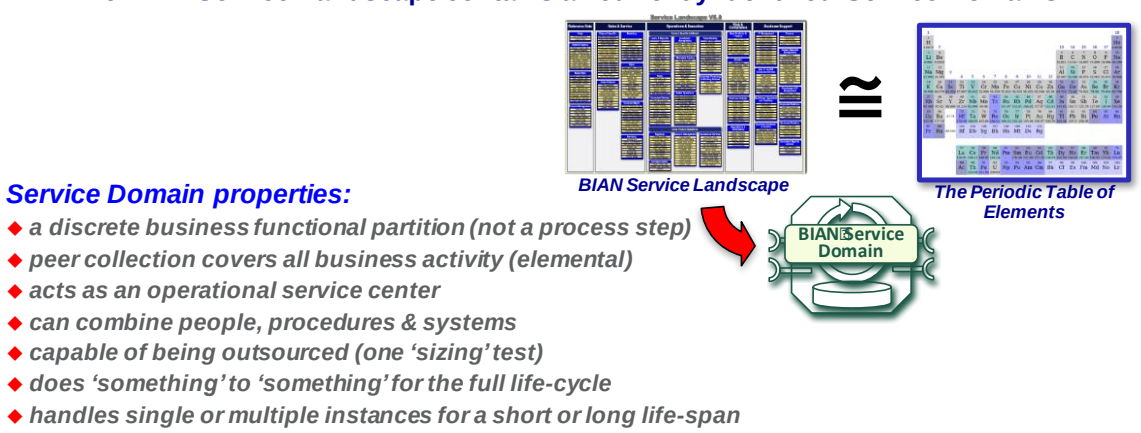


Figure 19 - Service Domain Key Properties

Though the operational characteristics of Service Domains span a very broad range, they all have the same basic design pattern – they each do something to something from start to finish as often as is required.

The way the many different BIAN design artifacts just described all link together to provide the overall Service Domain specification is captured in the BIAN 'figure 8' diagram shown below. Architects and developers will not normally need to reference most of these detailed artifacts for the Service Domains and service operations they use. These detailed design artifacts are used within BIAN to generate the semantic API specifications that provide the high level service operation descriptions that can be found on the BIAN Semantic API Portal.

The Figure 8 Diagram links the Design Elements

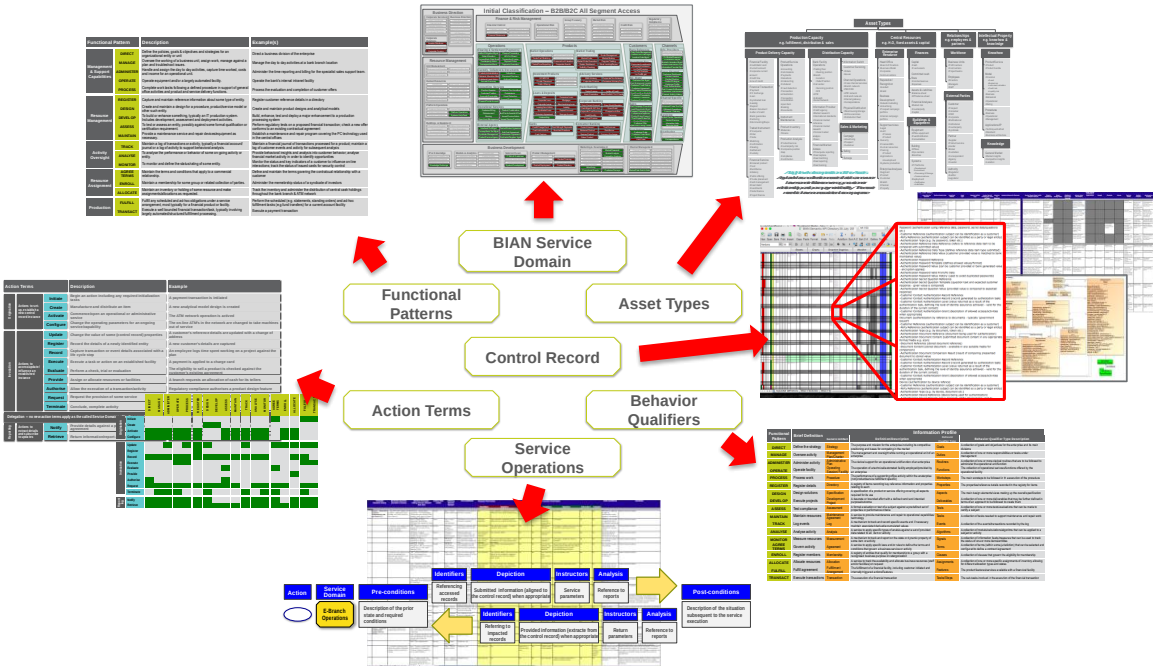


Figure 20 – The Figure "8" Diagram

3.3 BIAN Business Scenarios

To help explain the business role/purpose of the Service Domains BIAN provides various design artifacts that provide examples of their use. The first of these artifacts is the BIAN Business Scenario. A BIAN Business Scenario models the handling of a single

business requirement or event rather like a conventional business process. A business scenario is an archetypal illustration of a possible set of service interactions that might occur between a collection of Service Domains as they handle the event. A BIAN business scenario has the following properties:

- ***Bounded*** - It should have a clearly defined business goal/objective with an associated start and ending position
- ***Meaningful*** - It includes sufficient content in terms of the Service Domains and their service operation exchanges to represent a coherent example of business activity for a business practitioner to review
- ***Non-Prescriptive*** – It presents a sensible sequence/flow of interactions as an archetypal flow but this sequence and the thresholds/triggers for service exchanges are not mandatory, just viable examples
- ***Loose Coupled*** – though the scenario may read as a linked sequence of exchanges, this serial coupling is not imposed. A service link shown between two Service Domains in the scenario simply indicates that a service dependency exists between them in the context if this business event– how and when this exchange is implemented and any start/end dependencies are not defined
- ***Non-exhaustive*** – a scenario does not attempt to define all required/possible service exchanges. Its intent is to clarify some specific role/behavior of the selected Service Domains by providing an example. For this reason, it is usual when defining business activity in an area of interest to use a collection of several overlapping business scenarios.
- ***Non-redundant*** – once a specific exchange pattern has been captured in one scenario within a collection this pattern can be excluded from the other scenarios for simplicity (it can be cross referenced if needed to avoid confusion)

Properties that can be captured in a business scenario but that are generally avoided for clarity include:

- ***Conditional and Multi-path Flows*** – most business scenarios will not include conditional/multi-path flows. If there are different options to be defined these usually are better captured as multiple scenarios within the overall collection
- ***Second Order Exchanges*** – most business scenarios limit the ‘nesting’ of dependent service calls (i.e. where a called Service Domain is shown to depend on it making a further delegated service call to be able to respond). Nesting technically breaches the fundamental principles of encapsulation in service oriented design. But there are situations, particularly when modelling real-time customer interactions, where these properties need to be shown for practical implementation purposes as already described.

The business scenario model is effective as a discussion mechanism to define and agree requirements with business practitioners. Also to clarify the specific roles of

Service Domains and their service exchanges for developers, helping them establish the functional partitioning intended.

Business Scenarios have many similarities with a conventional process model. In the table below the earlier mortgage application example has been redrawn to show one viable sequence of interactions in a BIAN Business Scenario. The format is similar to a more traditional process model with the involved Service Domains identified as the key actors in each column. The key difference as noted is that the sequence of exchanges is not tightly coupled in the scenario – exchanges can be triggered as and when and there is no assumed start/end dependency implied in the scenario.

Mortgage application captured as a BIAN business scenario

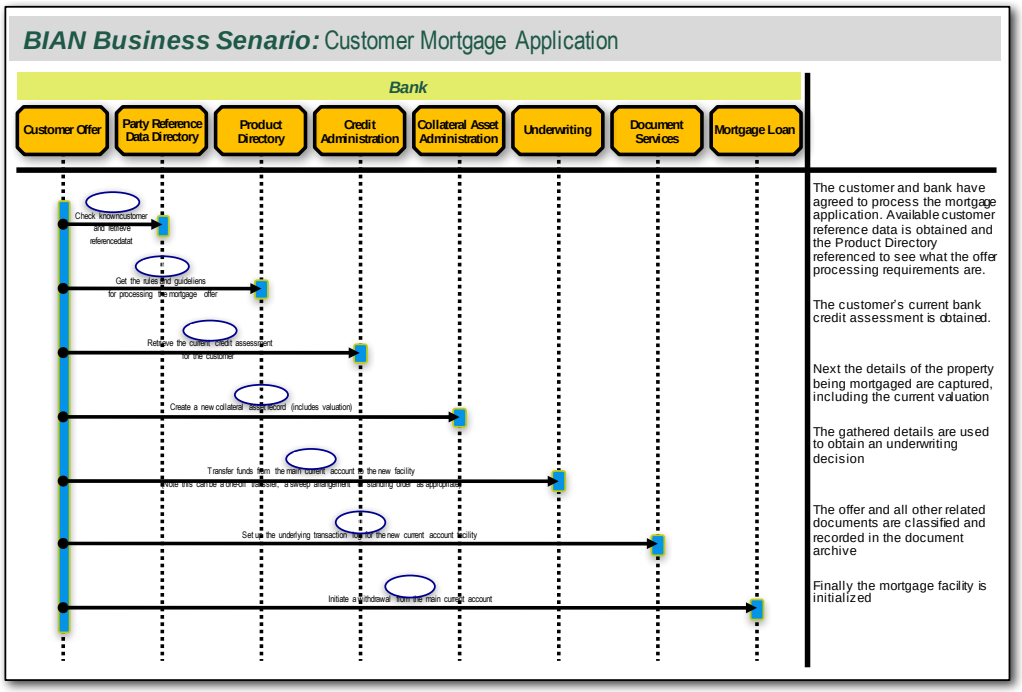


Figure 21 - Example Mortgage Business Scenario

Some aspects to highlight from the example:

- The involved Service Domains (the yellow/orange boxes) each have their own dedicated column. The archetypal service exchange flow runs from top to bottom

- The vertical blue box indicates the duration the Service Domain is active in the scenario
- The horizontal arrows indicate a service exchange/dependency between the Service Domains. The arrow points to the called Service Domain (i.e. the one providing the service that has been delegated to by the calling Service Domain)
- A circle at the calling end of the arrow indicates that the exchange represents both the call and the response (as is the case for all exchanges in this example – more complex scenarios can include nesting of calls with the response coming later in the service flow – see Figure16 earlier in this Section)
- The purple ellipse on a service exchange arrow includes the service operation's 'Action Term' – this property is fully described later – in essence the action term characterizes the nature of the service call (This notation can also include an additional behavior qualifier field when applicable)
- The service exchange text describes the purpose of the interaction in general terms in the context of this scenario
- The narrative in the column on the right outlines the overall flow of the processing from start to finish

When developing a new systems solution or mapping to one or more legacy systems, a collection of Business Scenarios is used with each addressing some particular event or business requirement of interest. As a guide fifteen to twenty Business Scenarios might be defined to cover the key requirements of a targeted business area (such as payments processing or customer servicing) though clearly the required number of scenarios greatly depends on the scope and complexity of the application design.

3.4 BIAN Wireframe

The BIAN Wireframe shows the available (first order) service connections between a related collection of Service Domains. A Service Domain may make use of more than one of the services offered by another Service Domain. (For example *request*ing a specific action be performed or simply *retriev*ing status information from the same Service Domain).

A wireframe is rather like a city map that shows the allowed service connection 'pathways' connecting the Service Domains. A business scenario is then one example of a journey that traverses this map using its particular service connections/paths.

A business scenario is referred to as a 'dynamic' model view because it details the behaviors/actions taken over time for some event. The BAIN Wireframe conversely is a 'static' model view of the Service Domains as it depicts their persistent available connections regardless of any timing or specific event/activity.

The diagram below is a simple wireframe perspective for the mortgage application scenario described earlier

The business scenario reorganized as a simple wireframe

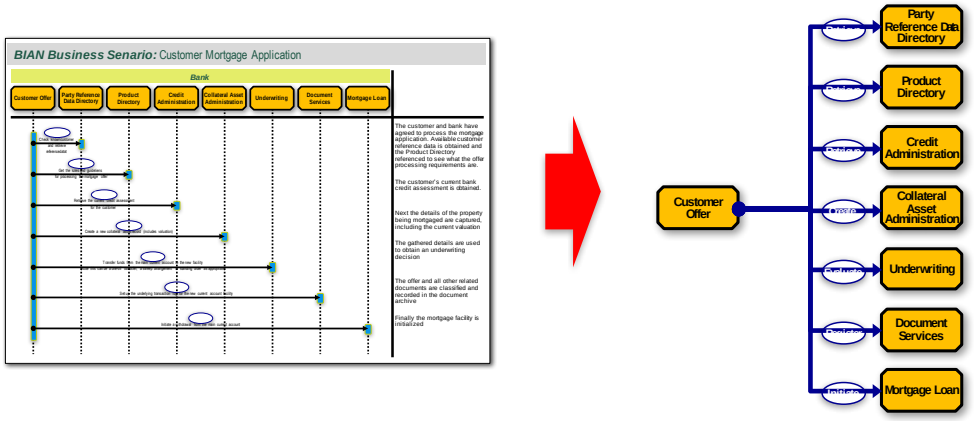


Figure 22 - Simple Wireframe for the Mortgage Application Scenario

As in the business scenario, the yellow/orange boxes are the Service Domains, the arrows indicate a service connection (pointing to the service provider from the caller) and the purple ellipses indicate the nature of the service exchange (referencing the associated “action term”).

Just as a collection of Business Scenarios is typically used to capture the main processing/event requirements for an area of interest. A wireframe is typically assembled including all of the Service Domains and service connections used in the same collection of Business Scenarios. Some additional connections and/or ‘boundary’ Service Domains can be added to round out the wireframe where helpful.
The key properties of the BIAN wireframe

- ***Non-exhaustive*** – the wireframe only needs to show available service connections that are used in the associated collection of business scenarios (some versions may show additional, even all available service connections, but these typically become too unwieldy)

- ***Arbitrary Scope*** – the selection of Service Domains and associated connections is informal. If it is necessary to add or suppress connections and or include/exclude Service Domains to clarify a particular viewpoint this is permitted

When a wireframe is assembled for a broad collection of Business Scenarios it can become quite complex. Arranging the Service Domains to avoid too many crossed

pathways can be a challenge and often it is necessary to strike a balance between including all required service connections and ensuring the readability of the diagram. As an example, the wireframe below covers general customer servicing activities. The specific Service Domains involved in the mortgage offer business scenario have been highlighted.

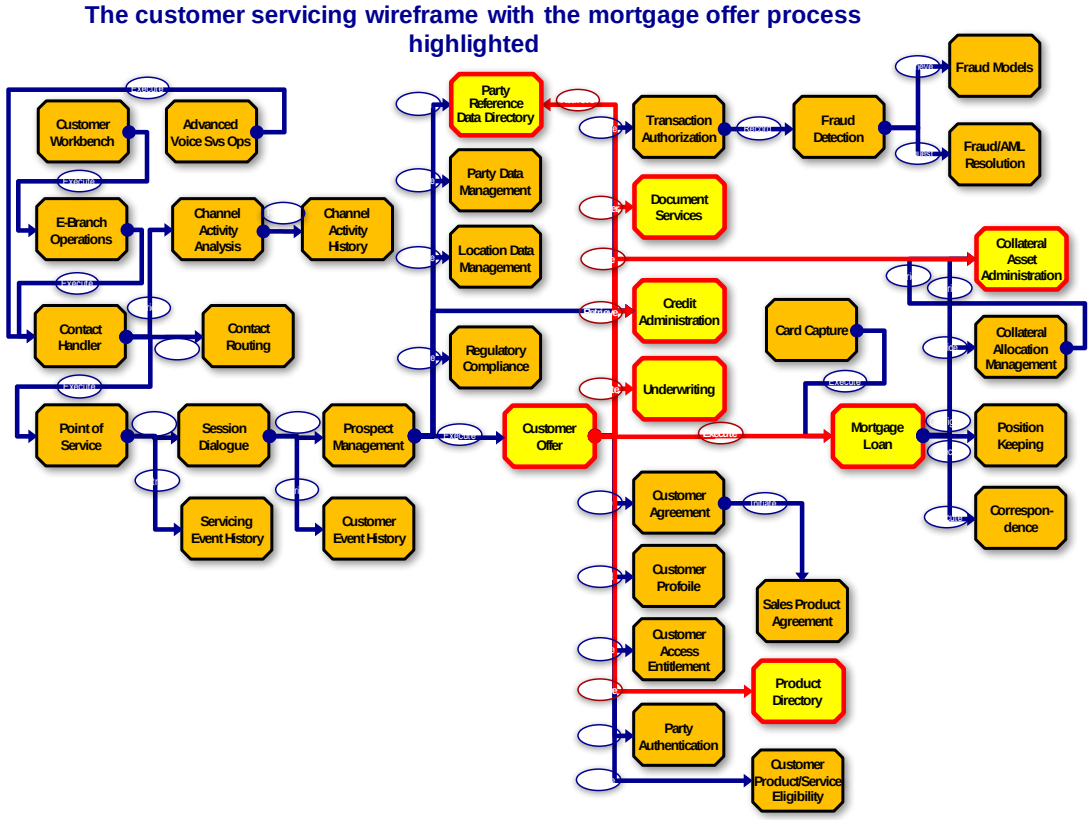


Figure 23 - Customer Servicing Wireframe with Mortgage Scenario Highlighted

For more complex projects a collection of wireframes can be used highlighting different areas and aspects of the development.

Combined scenario and wireframe views clarify the components for development

Having both the dynamic (business scenario) and static (wireframe) models of the area of interest is useful to fully understand the service-centered design for the technical leads and architects. In systems development the business scenario is important to confirm key business requirements are supported in a manner quite similar to process oriented design. The Wireframe then defines a stable blueprint over which the scope of

existing or to-be-developed applications can be mapped with the Service Domains corresponding to major functional partitions of the application.

With this overlay on the wireframe it is possible to map the interfaces that can be supported by API's between applications and the main internal exchanges within an application. This use of the wireframe is expanded on considering different technical environments later in this guide.

3.5 BIAN Semantic APIs (REST Mapping and the BIAN Semantic API Portal)

The BIAN standard is by choice implementation agnostic. But in order to support the use of the BIAN Service Domain partitions and service operations as a framework for container based architectures and the more general use of application program interfaces (APIs) the BIAN definition has been mapped to the REST architecture style. REST is the most popular approach being used for API development in the banking industry at this time.

For the purpose of mapping APIs to BIAN the assumption is made that the BIAN Service Domain matches the application boundary (the "A") of the API. The Service Domain's service operations then make up the collection of program interfaces (the "PI"s) that complete the API's description. In BIAN terms a semantic API consists of the collection of service operations offered by a Service Domain with the service operation specifications formatted in a manner that is suited for developer enhancement/extension (for example adding bank implementation specific reference attributes) for implementation in the REST architecture style.

Representational State Transfer (REST) has been developed specifically for creating Web services. It defines specific constraints that are intended both to ensure interoperability and efficient performance for applications communicating over the Internet. The REST approach provides access to 'resources' using a predefined set of stateless operations (stateless meaning that no client information is persisted at the service provider between service requests). The resource is identified with a URI and the service request will result in a response that returns values relating to the resource in the service payload. This payload can be presented in various formats – JSON being the most common (HTML and XML being popular alternatives). HTTP is the most common protocol used for the service request operations with the particular HTTP methods GET, PUT, POST, (PATCH) and DELETE being applied in the BIAN mapping.

REST Architectural Constraints

REST defines six constraints for compliance. The ways these relate to the BIAN design approach is summarized as follows:

1. ***Client-Server Architecture:*** *separation of concerns (no assumed link between client and server data)* – BIAN Service Domains can fully conform to a client-server architecture in implementation

2. ***Statelessness:*** *no client context is stored on the server* – though implementation agnostic, BIAN Service Domains support SOA design principles which can conform to statelessness where practical

3. ***Cacheability:*** *responses can support caching (handles sequential responses sensibly and responses are re-useable)* – again as BIAN is intended to support SOA design principles, selective cacheability can be fully supported in implementation

4. ***Layered System:*** *the client has no awareness of intermediary layers between it and the host* – as BIAN conforms to SOA design principles, encapsulation in particular, this constraint can readily be handled in implementation

5. ***Code on Demand:*** *the response can embed executable logic* – BIAN does not preclude that service exchanges can include executable logic, this typically being an implementation consideration for front-end applications

6. ***Uniform Interface:*** *comprises four more specific constraints: resource identification in request; manipulation through representations, self-descriptive messages, Hypermedia as the engine of application state –"HATEOAS")* – BIAN service operation definitions do not constrain the adoption of any of these service implementation features as might be appropriate

In summary the BIAN standard is a conceptual business model that defines service exchanges in semantic terms in a manner that is implementation and therefore also vendor agnostic. The supposition is that BIAN specifications will usually be deployed using service oriented architectural (SOA) approaches – BIAN specifically represents business activity with this goal in mind. SOA concepts in general align well with the constrains imposed by the REST architectural style and are not incompatible with a REST implementation in any significant way.

REST Archetypes

In the BIAN to REST mapping a Service Domain control record instance essentially represents the accessed resource. REST defines four resource archetypes (documents, collections, store and controller). To ensure the Service Domain control record is correctly interpreted it helps to align the different BIAN generic artifact types for the functional patterns to these four REST archetypes:

The REST archetypes for reference can be defined as follows:

- **Documents** – a singular resource concept. It is referenced using a conventional hierarchical naming structure:
  e.g. http://api.example.com/building-management/office-buildings/{building-Id}
  The state representation typically combines feature values of the instance

- **Collections** – represents a managed/directory collection of resources. The collection determines when to create a new resource instance on request.
  e.g. http://api.example.com/building-management/office-buildings

- **Store** – a client managed resource repository – it does not create new resource instances but enables a collection:
  e.g. http://api.example.com/cart-management/users/{id}/carts

- **Controller** – this resource handles a procedural concept. It acts like an executable function with parameters and inputs/outputs
  e.g. http://api.example.com/cart-management/users/{id}/cart/checkout

The different BIAN generic artifact types that define the control record make-up can be mapped to these archetypes as follows (note that only three of the four REST resource archetypes are actually needed for the mapping):

Functional Pattern Generic Artifacts Matched to the REST Archetype

Functional Pattern	Generic Artifact	Mapped RESTful Archetype	Example URI
DIRECT	Strategy	Document	http://api.example.com/enterprize-management/responsibilities/{strategy-Id}
MANAGE	Management Plan	Document	http://api.example.com/business-management/responsibilities/{management-plan-Id}
ADMINISTER	Administrative Plan	Document	http://api.example.com/administration-management/responsibilities/{aministrative-plan-Id}
DESIGN	Specification	Document	http://api.example.com/model-design/customer-models/{model-Id}
DEVELOP	Development Project	Document	http://api.example.com/application-development/retail-projects/{project-Id}
PROCESS	Procedure	Controller	http://api.example.com/back-office/payments/customer-billing
OPERATE	Operating Session	Controller	http://api.example.com/production/ATM-network/operation
MAINTAIN	Maintenance Arrangement	Controller	http://api.example.com/systems/computer/{id}/maintenance
FULFILL	Fulfillment Arrangement	Controller	http://api.example.com/consumer-products/current-account/{id}/arrangement
TRANSACT	Transaction	Controller	http://api.example.com/wholesale-products/equity-trade/{id}/transaction
ADVISE	Advice	Controller	http://api.example.com/wholesale-products/corporate-finance/{id}/advice
MONITOR	State	Controller	http://api.example.com/customer-relations/{id}/state
TRACK	Log	Controller	http://api.example.com/customer-relations/{id}/customer-history/log
CATALOG	Directory Entry	Collections	http://api.example.com/products-and-services/{id}/product-specifications
ENROLL	Membership	Collections	http://api.example.com/syndicated-loans/syndicate-members
AGREE TERMS	Agreement	Document	http://api.example.com/customer-relations/{id}/{agreement-Id}
ASSESS	Assessment	Document	http://api.example.com/products-and-services/{id}/{assessment-Id}
ANALYSE	Analysis	Document	http://api.example.com/customer-relations/{id}/profitability-analysis/{analysis-id}
ALLOCATE	Allocation	Document	http://api.example.com/issued-devices/device-type/{id}/{allocation-Id}

Figure 24 - REST Archetype mapping to BIAN Generic Artifact

Note the mapping of control record to REST archetype simply confirms that the BIAN Service Domain control record construct can be treated as a resource when accessed using the REST architectural style.

Service Operation to Endpoint Alignment

An obvious challenge when mapping BIAN to the REST form arises because the BIAN specification includes extensive references to actions and behaviors whereas the REST architectural style by definition exchanges only the accessed resource's feature and state information. In order to align to REST the BIAN service operation action terms that characterize the expected service response have been converted to their noun form. In this way the action is redefined as the result or outcome from the action being performed that can then be treated more readily as properties of a resource. The action terms and their amended forms when applied to REST endpoints are as follows:

Activate	*becomes*	*Activation*
Configure	*becomes*	*Configuration*
Feedback	*remains as*	*Feedback*
Create	*becomes*	*Creation*
Initiate	*becomes*	*Initiation*
Register	*becomes*	*Registration*
Evaluate	*becomes*	*Evaluation*
Provide	*becomes*	*Provision*
Update	*remains as*	*Update*
Control	*remains as*	*Control*
Exchange	*remains as*	*Exchange*
Capture	*remains as*	*Capture*
Execute	*becomes*	*Execution*
Request	*becomes*	*Requisition*
Grant	*remains as*	*Grant*
Retrieve	*maps directly to the HTTP GET method*	
Notify	*is not currently used in the BIAN mapping*	

To define BIAN Semantic APIs each default BIAN Service Domain service operation is matched to a 'REST endpoint' description. The scope/purpose of each individual BIAN Service Operation and its associated REST endpoint description is defined by three concerns:

- ***The Service Domain's core functionality*** – the most obvious consideration is the core business function performed by the accessed Service Domain itself. This functionality is best characterized by the Service Domain's control record. The service operation should be considered to act upon an identified control record instance (or instances) or some sub-partition of the instance

- ***The service operation action term*** – the action term refines the service definition by characterizing the particular purpose for engaging the accessed Service Domain. It results in access to and possibly updates to the attributes of the referenced control record instance (for example to *update*, *exchange* or *retrieve* information). Based on the action term the attributes of the resource (control record) are filtered to select those required for the input and output messages

- ***Optionally the Behavior Qualifier*** – is used to narrow the reference to a sub-partition of the referenced control record instance. As described earlier the behavior qualifier type is used to partition the control record into sub-partitions of equivalent operational properties to the 'parent' control record (for example a procedure is decomposed into its constituent work steps). Then as before, based on the selected action term the attributes of the resource, in this case the control record behavior qualifier partition, are filtered to select those attributes required for the input and output messages

BIAN Endpoint descriptions are far from implementation specifications

BIAN is an **implementation agnostic conceptual specification**. As a result, and as already described, BIAN only seeks to define Service Domains and their service operations to a particular level of detail. The level of detail is intended to be sufficient such that a user can switch between two service providers that both conform to the BIAN specification without significantly destabilizing up and down-stream business dependencies.

The BIAN Service Domain service operation descriptions that can be found on the BIAN Semantic API Portal are formatted to look like a REST endpoint specification only to ease their adoption by developers familiar with the REST architecture style. It is important for developers to recognize early on that these semantic descriptions are some way from implementation level specifications. The BIAN semantic endpoint definitions provide an unambiguous 'classification' or description of a business exchange that can be consistently implemented using the REST architectural style. To complete the physical implementation design a developer needs to add significant content as follows:

- ***The exchange may need to be further decomposed to finer grained endpoints to handle subordinate activities needed to support a practical orchestration/choreography for the interaction. This includes exchanges that might be needed to handle optional/advanced features, errors and exception handling***
- ***Detail and content must be added to the sematic attributes to develop the complete physical data specification for the message payload (mapped to existing host systems' data structures and/or applicable message data standards as necessary)***
- ***Additional attributes for operational/communication purposes (e.g. the message header, security/error handling, message indexing/numbering etc.)***

Standard design and development approaches and techniques are well defined for handling the above requirements. Any suitable combination of these approaches can be employed as appropriate for a specific implementation project to complete the REST endpoint physical implementation specifications.

The BIAN designs do not define implementation level detail. Actual implementation examples provided to BIAN by members are already and will continue to be reviewed to ratify and extend the BIAN semantic definitions where necessary. Examples of selected practical implementations may also be made available in the future. This reference implementation material will help in the adoption of BIAN by providing re-useable compliant development content.

The BIAN Semantic APIs are available on the BIAN API portal that is an open source site accessible through https://portal.bian.org/. The format of the endpoints as defined for the BIAN Service Domain APIs uses the following format:

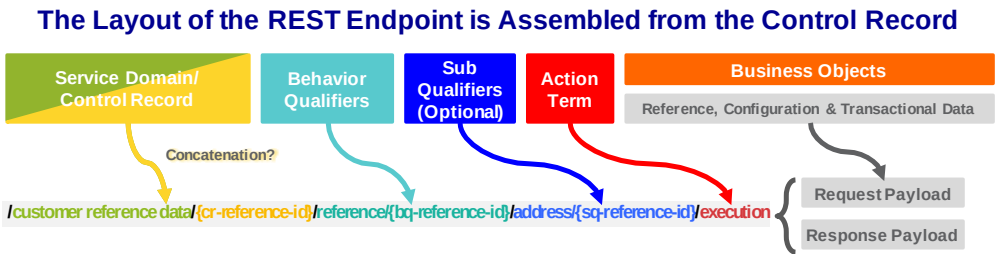


Figure 25 - BIAN API End Point Format

Note that the option to add sub-qualifiers is shown in the diagram. As noted earlier BIAN currently only defines behavior qualifiers to the first level decomposition and in many cases an implementation may need to add these additional levels of detail.

At the time of publication BIAN has developed Semantic API definitions for over 140 Service Domains. Additional definitions are being developed and constantly released to the portal. It is hoped that most development projects can already be supported by the Service Domains specifications currently available. The available coverage and plans for future delivery can be confirmed by accessing the Semantic API initiative on the BIAN website and if necessary contacting BIAN management directly.

The BIAN URLs provide a starting point for classifying and naming end points. Prefixes or postfixes can be used to link to mapped systems, version numbers etc. The adopted naming convention should be designed to facilitate searching through a service catalog. For example, sorting/filtering by host service domain can help with service identification.

3.6 Service Domain Event Triggering (Proposed design extension)

All of the design artifacts described in this Section to this point have been or are scheduled to support definition of the BIAN Service Domains. For more advanced service oriented architectures different ‘orchestration’ approaches can be considered for the developed applications.

In more conventional SOA implementations the service centers define discrete capabilities and the processing of business requirements/events is achieved through overlaying orchestration logic that coordinates the service calls between service domains. In this kind of implementation, the main benefit of service oriented design is that the operational capabilities are re-used. But the execution is limited to the pre-defined paths and new/changed behaviors typically require additional development effort. An alternative orchestration approach is one that is fully event driven.

In an event driven SOA the Service Domains have encoded dependencies that enable them to automatically ‘react’ between themselves to support any business requirement. When some business action or requirement updates the status of one Service Domain a series of events is triggered between related Service Domains so that each performs whatever actions are needed on their own behalf to fully address the business action.

In an event driven model the source business action results in an *asynchronous cascade* of triggered service exchanges that continue until all Service Domains reach a stable state reflecting the completion of all the necessary processing and updates required to fully handle the business action.

Sometimes the Service Domain state changes may need to be synchronized to take account of related start/end dependencies between the Service Domains. Other times the Service Domains can catch up in their own time (because the resulting actions and changes in their internal state does not impact any other Service Domain directly).

There are several design properties that need to be reflected in the Service Domain specifications to support a fully event driven design. Some example considerations are (in no specific order):

- ***Semantic Vocabulary Agreed to Required Precision*** – all exchanged semantic information must be defined as specific changes to the value of this information will often be a service triggering factor

- ***State Management & Service Triggering*** – comprehensive event profiles for the Service Domains and their service triggering logic. This should include configuring thresholds and policies. These can be linked to key information attributes or with control record and control record partitions as defined by the behavior qualifier type

- ***Service Operation Agreements and Policies*** – this includes more detailed service make-up definitions, including cross-referencing the policies and thresholds governing/triggering service exchanges as well as the required service performance, information integrity and security control features
- ***Transactional, Control, and Referential Exchanges*** – the required Service Domain information exchanges need to capture all transactional business activity, management command and control interactions and the background synchronization of shared reference information

- ***Defensive Operations*** – the Service Domain service operation implementation must always handle delayed/erroneous requests and respond in a graceful/defensive manner

- ***Exchanges must be Idempotent and Commutative*** – the Service Domains must handle duplicate exchanges and tolerate that any business event may result in parallel threads of activity that can complete in different relative sequences based on prevailing physical conditions

- ***Utilities and Middleware*** – to provide core Service Domain utilities such as a general service directory, data storage and management, transaction logging, data analysis and reporting, transaction assurance and state/trigger handling

- ***Routing/Communication Capabilities*** – to be able to discover and establish all required Service Domain connections with support for the associated message queue and event capabilities

To support an event driven model the current Service Domain information profile definitions will need to be extended to a lower level of detail and two additional properties will need to be built into the BIAN Service Domain standard:

1. ***Service Domain Events*** – a comprehensive definition of the state transitions and associated internal and external event triggers for the Service Domain. The events need to be defined at several levels:

    a. *for the overall operation of the Service Domain itself,*
    b. *for combined views of control record instances,*
    c. *individual control record instances, and*
    d. *behavior qualifier instances as appropriate, also*
    e. *potentially for selected individual attributes*

2. ***Service Domain Referential Dependencies*** – covers the patterns of access to information governed by one Service Domain that is referenced in the operation of other Service Domains. Note: this is not information that is exchanged in the usual course of transactional service exchanges, but is more likely to be exchanged and synchronized as a background activity through notification based service arrangements

BIAN will be coordinating with members to explore these requirements and the outcome will be reflected in later releases of this guide if necessary.

Section Summary

This section sets out the BIAN artifacts in sufficient detail for developers to reference the key properties behind the designs. Technical leads and architects are not expected to learn or become proficient in applying these specifications directly. Their primary source of design input is the BIAN Semantic API Portal and selected example BIAN business scenarios and wireframes that might relate to their specific development projects.

The BIAN artifact descriptions and explanations included in this section can be used by technical leads and architects to review the semantic API definitions, to determine how they have been defined where there may still be ambiguities in their meaning. They may also need to refer to some design artifacts if they wish to propose corrections and or extensions to the BIAN specification.

The artifact descriptions also provide a more general grounding in the BIAN approach for those that are interested to learn more.

4 – Implementation Approaches

> ***Note**: –BIAN has only recently published its semantic API extended specifications. It is early to define robust and comprehensive implementation adoption approaches. The emerging insights and techniques presented in this section are subject to revision as more practical experience is obtained.*

This final section describes how the component based BIAN Semantic APIs can be applied in development projects. Two main factors determine the context for a development project in this guide. The first is whether the target application addresses a back office, transaction processing type function or whether it supports a front office decision support and customer interaction type function. The second factor is whether the application adopts a conventional monolithic process oriented technical architecture or a more advanced container based, service oriented technical architecture. For the purposes of this guide the term monolithic indicates that the application employs a central shared/integrated database to support its application processing logic.

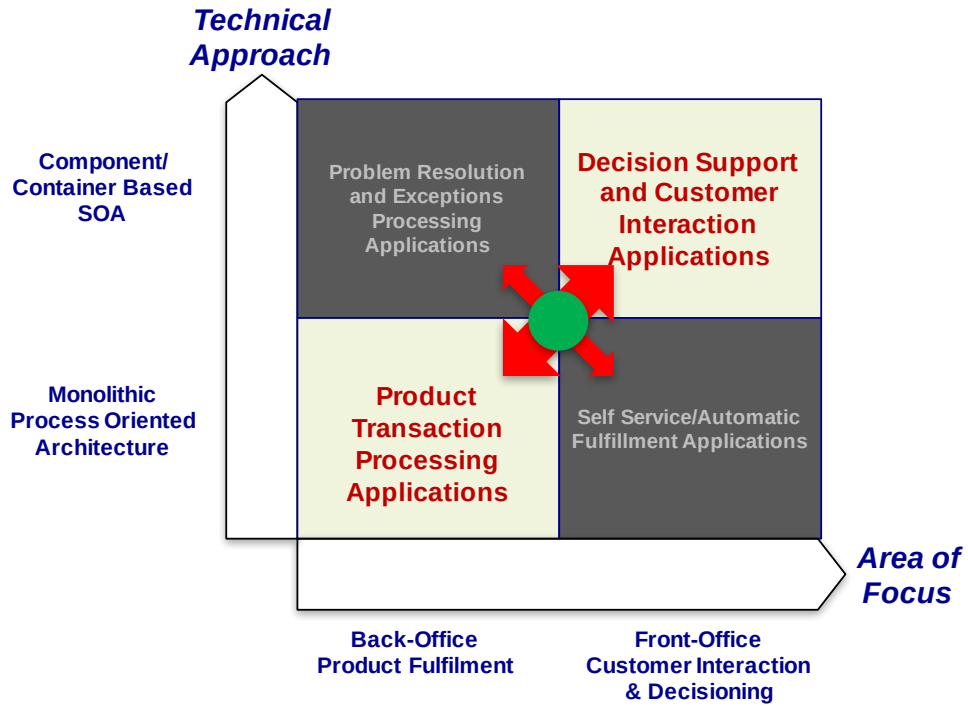


Figure 26 - Four Quadrants two Dimensions

As highlighted in the diagram most development projects will fall into two of the four quadrants shown. Back office transaction processing systems tend to adopt process centric designs and front office decision support systems can increasingly benefit most from container based service oriented designs. The approaches described here will focus on these two pairings. There is nothing to prevent developers from mixing and matching techniques as may be appropriate for their specific situation as the techniques are typically not incompatible.

In addition to different development contexts, some techniques are more relevant for 'green field' development and others apply where there is some degree of existing application re-purposing and/or integration. In practice most projects will combine both some legacy wrapping/repurposing and new development in varying proportions. Process oriented back office transaction processing developments typically include a greater portion of legacy repurposing. Conversely, most green field development opportunities will arise in the front office/container based application quadrant

The implementation approach covered by this final section of the guide is covered in three parts.

1 - Key Properties of Component Design – clarifies the key implications of adopting a component architecture for consideration by technical leads and architects. These include the application of component partitions, considerations for information governance and interfacing/communications approaches

2 - Adding Detail to the BIAN Service Domain Specifications – guidelines for interpreting and extending BIAN's high level semantic conceptual requirements down to physical implementation specifications. This describes the content and required additions at three levels: – conceptual requirements, logical designs and physical specifications

3 - Implementation Approaches – detailing some identified approaches to configuring physical designs that leverage the component model of business. This is an initial list of some possible physical configurations that are intended to address performance considerations and other issues. This list will hopefully be augmented as BIAN members provide feedback from actual deployment projects in the future.

4.1 Key Properties of Component Design

Component based design and development provides additional insights that can complement traditional process based designs. A component perspective allows the solution designer to define an application architecture that leverages specific component properties. In practice when dealing with legacy application in particular, architects may

need to deal with hybrid views that mix both process and component viewpoints, but for simplicity the properties of a pure component design are described here. These properties are described from three perspectives:

1. **Components & The Main Driver for Componentization** – discrete component based business functions support systematic operational capability re-use

2. **Information architecture** – contrasting component and process information handling approaches. An opportunity to improve information governance

3. **Communications** – components support standardizing and re-using service based interfaces

Each sub-section ends with a brief list of the main component related development considerations differentiating between those that apply specifically to back office process oriented versus front office container oriented developments. Note: that in this section the component concepts are described as they relate to a solution design regardless of whether the focus is on green field development or legacy renewal projects.

4.1.1 Components & The Main Driver for Componentization

Components define business functional building blocks, each representing the capacity to perform a specific business need. Any business application requirement can be supported by an appropriate collection of suitable functional components. As already described BIAN has applied specific design techniques to define a comprehensive and discrete set of discrete canonical business functional partitions particularly suited to a service oriented architecture (SOA). An architect should equate the BIAN Service Domain conceptual business functional components to major application functional modules. Where each Service Domain offers a collection of services associated with supporting a particular business capability.

When correctly designed BIAN Service Domain aligned solutions can be assembled to collaborate in any desired sequence and combination to support most banking activities. The discrete service centers can be engaged in many different business contexts supporting a very high degree of ***operational reuse*** as first described in Section 3.2.2.

The given example of operational reuse was a shared document handling service. It represents a specialized business function that provides services to capture, classify, verify, maintain and provide controlled distribution and access to documents. The example demonstrates how operational re-use is achieved when an enterprise establishes discrete specialized business capabilities that can be performance optimized and then shared across the enterprise. This enhances business effectiveness, flexibility and can reduce operational redundancy.

Operational reuse is not to be confused with the more conventional code-based ***utility re-use*** – where similar processing logic can be encoded and re-used. With utility re-use an often repeated functional requirement can be implemented using a programmed solution that is 'packaged' for repeated deployment. Utility re-use is highly desirable as it results in greater processing and information consistency and eliminated software development effort. But its benefits lie primarily in the code development productivity and quality of the application software.

Experienced technical leads and architects will be familiar with the potential savings from software utility reuse. The opportunities from operational reuse are less obvious but no less significant. The diagram below includes a simple schematic of a stand-alone consumer loan application on the left. A sample of its constituent service center components has been overlain (this is an informal selection of components only). On the right a consumer insurance application is also shown. The diagram shows the service center components that could be re-used and those that are likely to be unique to each application (i.e. have limited potential for operational for re-use).

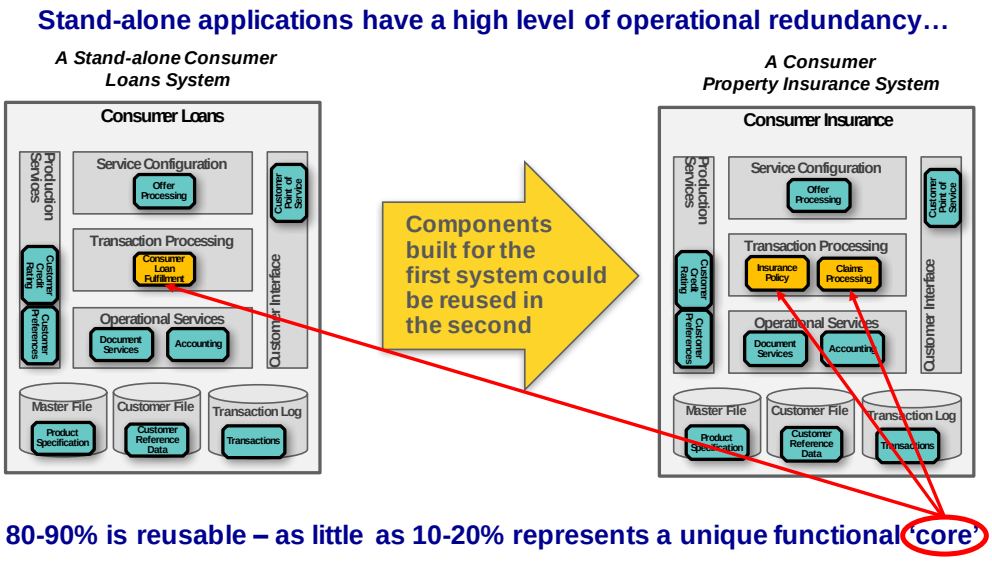


Figure 27 - Example of a Stand-alone Application and Operational Reuse

The example is typical of most stand-alone business applications - 80% or more of the application logic found in most can be a candidate for operational re-use when correctly engineered using a component design.

General Development Benefits:

Contrasting component design with more traditional process/monolithic designs with their more arbitrary partitions, a component based architecture has some general benefits:

- Clear partition boundaries to define and assign responsibility for discrete functional modules
- Defines elemental capabilities that simplify the selection and integration/assembly of solution modules/components
- Support for localized specialization/optimization of individual components (this can include their technical/architectural features)
- Support for different sourcing options for solution components

Benefits for Back Office Process Oriented Developments:

Components can be 'hard wired' together for back office transaction processing to ensure throughput performance. Components then operate as a more tightly coupled factory processing platform. Benefits from component insights/partitions include:

- Identifying boundaries and opportunities for batching/scheduling/decoupling linked activities along the production transaction processing flow
- Load balancing processing activities between component partitions
- Developing standard external access/reporting interfaces for information requests from outside the main transaction stream (these service requests would be aligned to the matched Service Domain operational services)

The patterns of connections are likely to be fairly stable and so connections can be more permanent in nature with limited disruption – service enablement has only limited application in the back office generally.

Benefits for Front Office Container Oriented Developments:

Components can be service enabled for the more interactive and decision support related activities found in the front office. The component architecture supports the development or applications that work as a loose coupled collaborative network. Key benefits from component insights include:

- Support for effective operational reuse of front office business functions
- Supporting the ability to implement flexible combinations of business activities for different operating/business models
- Ability to engineer and optimize multiple concurrent asynchronous processing interactions and exchanges between front office activities

4.1.2 Information Architecture - Contrasting Component & Process Approaches

The most significant difference between a component based and the more traditional process oriented application design is the way the business information is handled within the application. This is particularly the case with the specific design approach BIAN has used to define the Service Domains – the *asset leverage* model as described earlier in this guide.

The Service Domain Component Information Architecture

With the asset leverage model as already described, each BIAN Service Domain has a standard structure – it applies a specific pattern of behavior or control (functional pattern) to instances of some type of asset for the complete lifecycle, every time the business requires it to do so. For example, the Service Domain Customer Relationship Management applies the 'management' control pattern to instances of a 'customer relationship' (an intangible asset) for the duration of their relationship with the bank and it does so for every bank customer.

The component design adopted by BIAN has a number of implications for a Service Domain's information management:

- **Persistence** – the BIAN Service Domain defines a persistent business capability with its associated information store (database) – it may be active or inactive at any point in time, but it can always be available to respond to external service requests and typically also executes its own internal schedule of actions

- **Fully Encapsulated/Autonomous** – because the Service Domain is responsible for the complete life cycle operation of its business role it consequently governs all of the associated business logic and information required to perform its responsibilities for its complete lifespan.

- **Discrete/Non-overlapping** – each Service Domain is defined to perform a single discrete and unique business function. It may delegate actions through service calls to other specialized Service Domains. But the Service Domain is accountable for the outcome of all delegated tasks and the interpretation of any returned information.

As a result of these design properties all enterprise business information can be ***uniquely assigned to a single governing Service Domain*** where it is maintained for its complete lifespan. The information exchanged through service operations provides

the values/status details of information governed by one Service Domain that can be interpreted and applied to information governed by another. But each Service Domain maintains its own complete and independent information viewpoint and is responsible for the integrity of its own governed information.

An example highlights component based information governance

Consider how the home address details maintained for a bank customer may be referenced in many different business contexts. The governing Service Domain for this information is the Party Reference Data Directory Service Domain. In this case its governing responsibilities could include verifying the accuracy of the value provided (is it a real residential address? does the customer actually live there?) and ensuring that it is kept up-to date (has the customer moved or has their accommodation status changed in any way?). The Service Domain does not only have to maintain the information value but also must be able to qualify the integrity of the information when its value is provided to any other Service Domain. This is so that the calling Service Domain is able to determine whether the provided value is fit for its own specific business purposes.

This service dependency/arrangement can be clarified using some example references made to the customer's home address as governed by the Party Reference Data Directory Service Domain and made available to others by calling its offered 'retrieve' service operation:

- **Correspondence** – the Correspondence Service Domain needs to determine the address to send bank missives – this can include a mailing address (and other media locations such as email, a cell phone number for texts etc.). It may determine that the residential address value provided is good enough to use for its mailing address. Though it may adopt the same address value, the mailing address is in fact a different business information concept. For example, Correspondence may allow a customer to define a temporary mailing address (say when they are away on holiday) that would override the referenced home address value for a period.

- **Customer Agreements** – the Customer Agreements Service Domain needs to determine the legal residency (address) that it applies to the jurisdiction of contracts it holds with the customer. It may use their given residential address as the initial value for its research. But in most cases it will need to perform additional checks to adequately verify their residency status (such as requesting additional documentary proof)

- **Collections** – the Collections Service Domain may need to retrieve collateral against a failed loan. For this it could need to provide the last known location details for the collateral to a 3rd party collections agency. It could simply use the customer's reference home address held on file. But given the sensitivity it should probably verify that this is the most up to date version by requesting that the value is checked/refreshed before passing it on

- **Current Account** – the Current Account Service Domain may need to refer to the customer's home address to print on issued checks as a customer reference. The value of their home address maintained on file is likely to be completely adequate for use as their printed check owner's address information.

The different references clarify that even when several Service Domains maintain information that has a common data type/form (e.g. an address) and may have the same value at any point in time that the business context and purpose for the information is unique for each Service Domain. A Service Domain must determine whether it is appropriate to use the value for an attribute governed by another Service Domain for its own purposes.

The fact that a Service Domain is responsible for the full lifespan governance of the information is important to underscore – as noted every Service Domain handles the control pattern applied to the asset instance from start to end. As a result, it must handle/oversee the initial capture/verification, maintenance/updates, support any reference to and then undertake final deletion/archiving of all of its governed business information.

When implemented effectively the BIAN component approach can support the definition of a common business language across the enterprise. Furthermore, the business information and information exchanges are defined within an explicit business context. This is provided by the business definition of the governing Service Domain and the particular interpretation of the shared information applied by any calling Service Domain.

The Process Information Architecture

The process information architecture differs significantly from the component information architecture. The process view captures a dynamic business event that occurs at some point in time. It details a linked sequence of associated tasks that are triggered in a dependent series. There are typically a finite number of allowed paths through a process based system to take account of different processing variations and optional steps, but these are limited to keep things manageable. The end-to-end processing logic is supported by a shared database that provides access to any business information that is

created, referenced, or updated (and archived/deleted) throughout the event's processing.

Though the collection of business information closely matches the scope of information used in the equivalent component model view, the process model imposes no specific organizing mechanism to reconcile where the lifespan governance of the information is handled. In the process model an inventory of all accessed information is made and a database design created to optimize the way this information can be accessed throughout the specific processing patterns supported by the application.

In some cases, it might be obvious where some information is most sensibly sourced from an external facility where it is managed collectively for all users (such as a central customer information file – "CIF"). But in a monolithic process oriented implementation much of its information will be maintained independently in its local integrated and purpose optimized database.

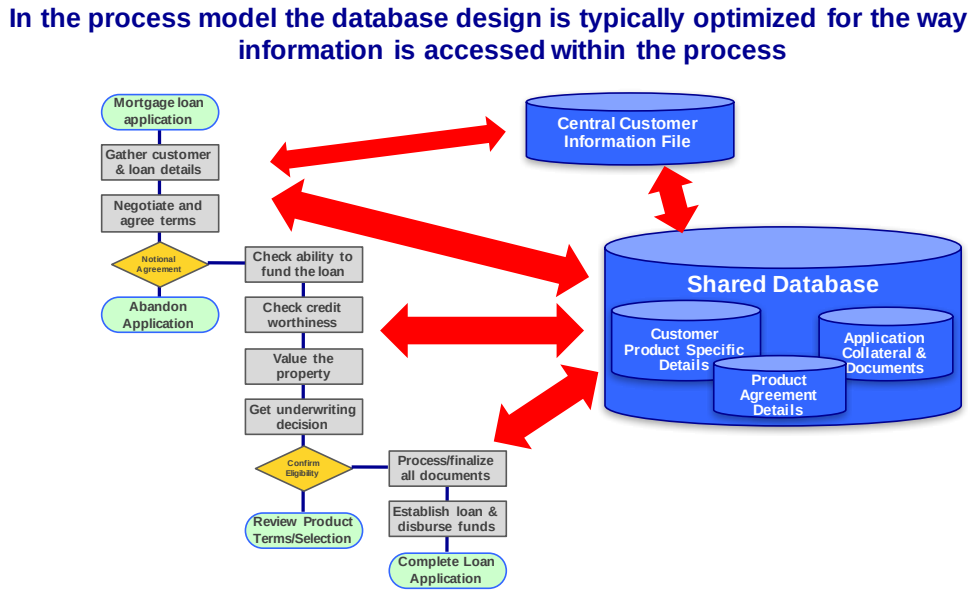


Figure 28 - Database Related to the Process Model View

Due to the complexity of most banking application portfolios it is extremely difficult to trace the shared information dependencies between applications. Other than shared central information such as the CIF there will probably be limited opportunity to identify and synchronize with other users of the same business information handled in applications elsewhere. This leads to inevitable business information duplication and fragmentation.

The table below shows how the process model accessed business information can be mapped to the Service Domains that would be needed to handle the same business event.

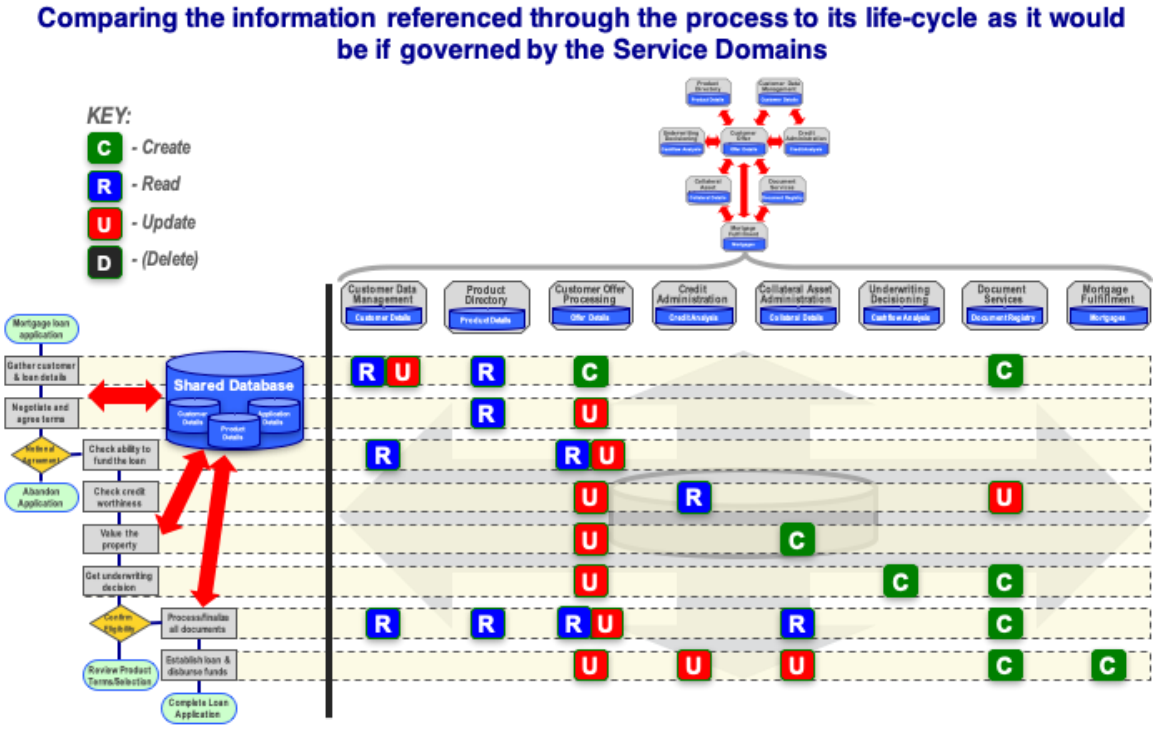


Figure 29 - Process CRUD linked to Service Domain Information Governance

The list below considers how the process mapped information is accessed and where appropriate considers the better governance opportunities that can arise when a component oriented design is applied (with life-cycle Create/Read/Update/Delete access indicated):

Customer Data Management (R,U) – it is likely that the process application will access a central customer information file (CIF) for some customer data, but it may augment this with additional locally managed information specific to the mortgage offer if the CIF content is limited. In a component design the customer reference data should comprehensive and support any retrieve and update requests that are made during the offer process

Product Directory (R) – it is less likely that a central directory detailing the specification of all products in a standard format is available and so the process

application is likely to embed its own local version of the mortgage product features within the offer process – such as pricing, eligibility and required documentation. In the component design there is a single place to go to maintain all product specifications (the Product Directory Service Domain) – when the product specifications are replicated and fragmented across multiple processing application they are clearly much harder to maintain

Customer Offer Processing (C,R,U) – as noted with the previous product directory comment, mortgage product specific offer requirements are likely to be built into the process application. This will result in duplicated offer processing logic for different products. With a service based product directory that can store the different product offer requirements it would be possible to build a single reusable generic offer process capability that configures its actions according to the selected product. It interesting to note that the process application handles the complete lifecycle of the offer information in this example (D-delete can perhaps be thought of as the archiving of the completed offer) – The scope of the process function maps most closely to the associated customer offer processing component

Credit Administration (R) – this is likely to be an established shared service in most banks, hopefully with a well-designed interface that enables its easy integration with the process application. In this case it maps directly with the component design

Collateral Asset Administration (C,R,U) – it is less likely that a general purpose collateral administration service is available and so mortgage specific collateral handling is likely to be built into the process application. This will make it harder to assemble a consolidated view of the customer's collateral position across multiple products

Underwriting Decision (C) – this like credit administration will hopefully be an established external service that can be sensibly interfaced with the process application. This is another case where the process view is likely to match the component design

Document Services (C) – this is a good candidate for a shared service capability. For most banks however the initial investment required to establish a documents services unit can be prohibitive and each process oriented application ends up keeping track of any documents required and created in isolation for the specific product it supports. The resulting fragmentation is a significant cause for errors and inefficient processing. For example, annoying repeated requests for the same documents being made to customers

Mortgage Fulfillment (C) – the offer process simply initiates the set-up of the new mortgage when appropriate in this example. It is most likely this is done by contacting a separate mortgage fulfillment application. Much of the information gathered throughout the offer process needs to transferred as the new product is initiated

As the examples clarify, the practice of building a local, high performance process oriented databases can result in fragmentation of the enterprise's business information. It is arguably one of the primary sources of complexity in legacy application portfolios with widely overlapping process solutions. The component design can highlight the business information governance requirements clarifying where information is best created and maintained through its lifespan.

Contrasting the Potentially Conflicting Issues of Performance and Consistency

The component based and process information architectures both have specific strengths and weaknesses. Many of these may be leveraged or mitigated with different application design and implementation techniques. At the conceptual level the differentiating properties are:

- Component Information Architecture Strengths
  - all business information governance is uniquely assignable to a single responsible business entity
  - the business context for information is well defined. Avoiding the incorrect inference that similar types of information used in different business situations must always share the same information value
  - the complete life-cycle of the information can be managed, ensuring appropriate action can be taken to maintain the integrity and currency of the information throughout its usage

- Component Information Architecture Weaknesses
  - providing access to singularly governed information introduces the potential for delay/latency and possible access limitations/constraints (during information updates in particular).

- Process Information Architecture Strengths
  - business information is defined to support the processing logic precisely
  - business information can be structured to ensure highly efficient access throughout the process
  - common enterprise reference business information can be easily duplicated and integrated where available

- Process Information Architecture Weakness

- local business information views fragment the overall enterprise model and can lead to extensive processing and data inconsistencies
- designs may not be readily adaptive to changes and enhancements

Benefits for Back Office Process Oriented Developments:

The process information model is most likely to suit the back office where information access performance can be optimized. The stability of the links and boundaries between processing applications and the narrow focus of the information they reference mean that reconciling local and shared views of the business information can generally be handled within the monolithic information architecture.

The component governance model may be useful to help reconcile common information associations between applications to limit the process and information fragmentation in legacy applications. Furthermore, as processes are subject to change and commercial solutions may wish to support different process configurations, a component design can provide greater stability over time and be more flexible to meet different processing arrangements

Benefits for Front Office Container Oriented Developments:

The component information model is most likely to suit the front office. A far wider range of information sources and services are likely to be accessed. The governed information model supports the flexibility to make connections when needed and in any combination. As the exchanged information is managed autonomously by each Service Domain its integrity can be assured as necessary. Clear information context, definitions and properties can also ensure that exchanged information values are correctly interpreted across the business.

4.1.3 Communications – Component Support for Standard Services

The BIAN standard has been expressly defined to support the adoption of a standards based service oriented architecture (SOA). There are two features of the Service Domain specifications to highlight in this regard:

- BIAN Service Domains define discrete and elemental business capabilities
- BIAN Service Domains handle the full life cycle occurrences of their specific role

As a result, the BIAN standard can be used to define a comprehensive and non-overlapping set of service exchanges covering all banking activity to a certain level of specification detail. Furthermore, as the Service Domains and their associated service operations are canonical (consistently interpreted across all implementations) – the

resulting underlying service exchange specifications can also be applied as an industry standard.

In contrast, for monolithic process type applications the communication interfaces between applications are more likely to be point to point (i.e. unique and dedicated to each application to application exchange). Each interface will usually need to handle specific features imposed by both involved applications. When building new application interfaces developers usually attempt to re-use existing interfaces to minimize development effort. In practice however, other than for the most utility types of exchange, the required adaptations and enhancements often result in the definition of a new (and overlapping in terms of repeated capability and content) interface.

The proliferation of overlapping point to point interfaces is another factor that contributes to the complexity and severe fragmentation found in most banks' application portfolios. When a component architecture is established it is possible to start 'standardizing' exchanges by implementing standard services and service based communications.

The BIAN Service Operations are High Level Specifications

The Service Domain and service operation definitions are presented at a conceptual level and described in semantic terms. What is intended from the level of detail provided is that the nature or purpose of the service exchange can be consistently interpreted between deployments.

The intention is that a bank or solution provider that aligns to the overall BIAN model can switch out the provider of a service for an alternate service provider without destabilizing other aspects of their business operation. For example, a bank that makes use of an external service provider to provide credit reports on individuals should be able to switch to another compliant service provider without having to rework their entire customer management function.

It should be expected that in switching between service suppliers there will be some low level interfacing and mapping work to do to re-establish the physical connection, but the key business information and service requirements should be supported. Finding the right level of detail for the BIAN semantic specifications is a practical challenge and something that has to be refined in practice:

> *When BIAN Specifications are too high level* – the precise business purpose for aservice exchange is ambiguous and service subscribers switching between providers will find that processing anomalies permeate beyond the immediate service interface and might start to destabilize other aspects of their business

When BIAN Specifications are too detailed – the specifications will impose processing constraints and requirements that restrict service providers' ability to conform to the standard. In addition, with overly specific service definitions, service providers may not always be able to cleanly align their services to the corresponding BIAN service

Finding the right level requires judgement and is likely to be something that improves based on practical experience. The two specific features of the BIAN standard listed above are intended to facilitate this.

In addition to providing high-level business application designs, the Service Domains can be used to structure organizational and operational aspects of the enterprise. The discrete role of a Service Domain can be used to define a specialist operational business function/service that can be assigned to a particular organizational unit and provide shared services across the geographic layout of the enterprise.

BIAN Service Domains Defines Discrete and Elemental Capabilities

Each Service Domain performs a unique and discrete business purpose. The scope of the service Domain is also defined at a level where this business purpose is elemental in nature, meaning that it can only be adopted in its entirety.

As a result, the service operations that access any one Service Domain have a very clear and concise business purpose that can't be confused with the role of any other Service Domain – there should be only "one place to go". The associated types of information and actions that can be accessed by the service operation should be readily associated with the Service Domain based on its specific business role.

The business role of the Service Domain is also best characterized by its *control record*. As a result, any offered service should also be directly relatable to the control record in terms of accessing its governed information or invoking some associated function that it handles.

BIAN Service Domains Handle the Full Lifecycle of Their Role

Because every Service Domain handles is specific business role from end to end, a standard collection of service operation types can be defined to handle all associated states and reporting perspectives. The 'action terms' defined by BIAN characterize the different types of service operations that can access any Service Domain.

Furthermore, as described earlier in Section 3.2.4, the BIAN design breaks down the Service Domain's control record into constituent parts using behavior qualifier types. This mechanism is used to ensure that the same action term can be consistently applied to the Service Domain's control record in its entirety or on some sub-partition of it. This sub partitioning provides increasing focus as might be required to isolate a more specific service requirement.

General Development Benefits

The general communications related benefits from a component architecture include:

- Service Domains define the discrete and assignable sources and consumers of service operations
- Service Domains define a clearly bounded scope of specialized business activity to narrowly define the meaning of all associated offered services
- The Service Domain's control record (and its behavior qualifier based sub-partitions) defines the governance context for all exchanged business information, ensuring overall information integrity

Benefits for Back Office Process Oriented Developments:

The component architecture defines standard service boundaries between components that can be implemented as high performance point to point interfaces if needed to support the main transaction flow.

Standard services can be also used to provide structured reporting access to the back office applications, in particular to support the information extracts made to the front office applications. The use of standard reporting extracts based on component designs is a key tool that can be applied in legacy application re-purposing.

Benefits for Front Office Container Oriented Developments:

The component model is specifically suited to the service enabled operations of loose coupled front office applications. The component model is a critical enabler for implementing an effective service oriented architecture (SOA).

4.2 Adding Implementation Detail

The BIAN standard defies the mainstream functions and exchanged business information at the level of conceptual requirements. These high level specifications need to be expanded to much finer levels of detail to support development projects. Three standard levels are considered:

Conceptual Requirements – is the main level at which the BIAN standard is pitched – defining the business operational functions in a component architecture

Logical Designs – some of which are partially defined within the BIAN standard consider different implementation approaches and options

Physical Specifications – details the code logic and data needed to implement the designs in practice (only very limited insights are provided in this guide at this time)

In this section guidelines are provided as to how to interpret the BIAN standard content and extend it as necessary down to implementation level physical specifications.

4.2.1 Conceptual Requirements

It is worth noting that the conceptual Service Domains do not actually represent the top level of a complete enterprise business design. Business planners/strategists can exploit an additional layer or perspective above the conceptual Service Domain partitions. This layer defines the 'value view' of the business. It details different business interactions and motivations (that can invoke the Service Domains when appropriate) and associates business value creation and more general business performance measures with the outcomes.

The business value layer can be leveraged for a broad range strategic planning and enterprise investment decisioning activities. BIAN's work developing its Business Capability Model is intended to link the BIAN Service Domains into the business value analysis layer (this on-going work can be reviewed at BIAN.org).

For most BIAN users and technical leads and architects in particular the less abstract business functions performed by the BIAN Service Domains and defined at the conceptual level provide the best entry point. The BIAN Service Domains each represent the capacity a business can possess to perform a specific and discrete business function. The Service Domains can be treated as the major building blocks for assembling banking application designs. At the conceptual level the Service Domains define this capacity in terms of the business requirements it addresses – i.e. 'what' the

Service Domain should be able to do without stipulating in any way 'how' the Service Domain might perform this function.

For architects reviewing the Service Domain definitions at the conceptual level the goal is to identify discrete functional partitions that can correspond to major functional modules in their target application development. Because of the Service Domain designs these functional partitions can be implemented as autonomous container based service centers when appropriate.

As noted earlier in this guide (Section 2.1) the motivation for adopting Service Domain partitions is that it results in a modular/component application architecture with key properties that include briefly:

1. **Defines components that support operational re-use** – each Service Domain matches a discrete business function. It encapsulates its business information and logic such that it can be implemented and deployed as a reusable operational service provider. (Note that only 20% of the BIAN Service Landscape covers product specific processing with limited potential for operational re-use. The remainder represents highly reusable cross product operational activities)

2. **Supports incremental development and adoption** – Service Domains define what is done, not how it is done internally. When properly engineered an aligned application can be developed and adopted incrementally across the enterprise.

3. **Canonical specifications** – BIAN Service Domains define generic functional building blocks that make up any bank. Aligned industry solutions should be interchangeable hopefully with only limited and localized mapping/reworking.

BIAN conceptual Service Domains definitions can optionally be combined with selected BIAN business scenarios and wireframes to provide example business context. Together this clarifies the purpose and boundary of the Service Domains to provide a robust definition of major partitions that can be reflected in the application design to support a component implementation. The key insights the solution designer should take from the conceptual Service Domains as they set out the overall structure of their application design include:

- The core business role/function supported by each Service Domain partition
- The type of business information the Service Domain governs
- Representative service operations offered as major application partition interfaces
- From associated scenarios and wireframes an indication of any delegated service dependencies

As stated, at the conceptual level solution architects/designers should be aware that the BIAN requirement descriptions are limited:

- They only describe general/mainstream requirements,
- They do not address errors and exception conditions
- They do not consider any non-functional properties such as performance and security

Conceptual Business Information and ISO20022 Mapping

The information governed by the Service Domain is presented in semantic terms and defined at a conceptual level. The provided name and descriptions of the information attributes are intended to be representative and intentionally avoid any implementation specific formats (other than as examples). For example, an attribute might refer to a “product instance reference” meaning that the attribute should uniquely identify a particular occurrence of an in-force product. The name and description avoids suggesting any specific naming convention or format for the information attribute as this is considered to be implementation specific.

BIAN attempts to match its Service Domain information attributes to existing industry conceptual object models. Currently the dominant prevailing standard is the ISO20022 Business Model. Where possible BIAN currently maps its semantic attributes to the ISO model, but due to gaps and misalignments it has been necessary for BIAN to maintain its own intermediate Conceptual Business Object Model (the BIAN BOM).

Conceptual Requirements Level Summary

The BIAN conceptual designs are intended only to provide sufficient detail that aligned developments will adopt standard application modules/boundaries that support component based development and improve general interoperability. Where appropriate the component based application designs are highly suited to the adoption of SOA implementation approaches.

The mapping to the industry standard ISO20022 Business Model is intended to assist with the consistent interpretation of the business information. This recognizes the current practical limitations in the available industry standard information specifications.

4.2.2 Logical Designs

The logical designs provide the next level of definition. In essence they address the ‘how’ underlying the Service Domain’s conceptual requirements. At this level the solution architect can expect to add significant detail to the Service Domain descriptions. The logical designs will quickly start to include site or implementation specific details as the

descriptions are extended. BIAN lists general guides for logical design options that can be adopted, but avoids defining extensive detail in order to remain implementation independent. Before describing the different aspects of the logical design that can be applied, the BIAN technique of Service Domain 'externalization' is explained.

Externalization

As frequently described already the role of a Service Domain is to apply a pattern of behavior to instances of a specific asset type for the complete life-cycle. This cycle defines the Service Domains functional scope of responsibility. The state of the subject it acts on as it completes this cycle is tracked using the control record and this holds the key business information governed by the Service Domain. In fulfilling its core business function, a Service Domain will almost always need to access the specialist functionality (and the associated business information) handled by many other Service Domains. Defining the correct boundary between a Service Domain's own responsibilities and those that it 'delegates' to other Service Domains through service calls is called 'externalization'.

Determining when functionality or business information belongs within a Service Domain or should be external and accessed by delegated service calls is critical to ensure that Service Domains remain discrete/non-overlapping and that they are elemental in their business role (as described in more detail in Section 3.2.). Applying this design consideration is a key aspect of the work performed by the BIAN Working Groups that define and ratify the published BIAN Service Domain designs.

Revisiting the externalization decisions underlying a Service Domain can be useful for technical leads and architects to better understand its business role and can also help when mapping Service Domain partitions to legacy applications. Determining whether a function or associated business information is contained within a Service Domain or should be 'externalized' and accessed through a service boundary boils down to a single test:

> ***Is it (the considered function or information) sensibly considered a feature of or property of the Service Domain's control record instance, and can it only meaningfully exist as an aspect of that control record and its life cycle? Or does it refer to some other distinct entity with its own independent lifecycle, that is governed by its own specialized Service Domain and handled as a property or feature of its control records?***

In the earlier example of a mortgage application one Service Domain: Customer Offer delegates several actions to other Service Domains. By considering the life cycle of their respective control records the associated externalization decisions can be readily understood:

Mortgage application captured as a BIAN business scenario

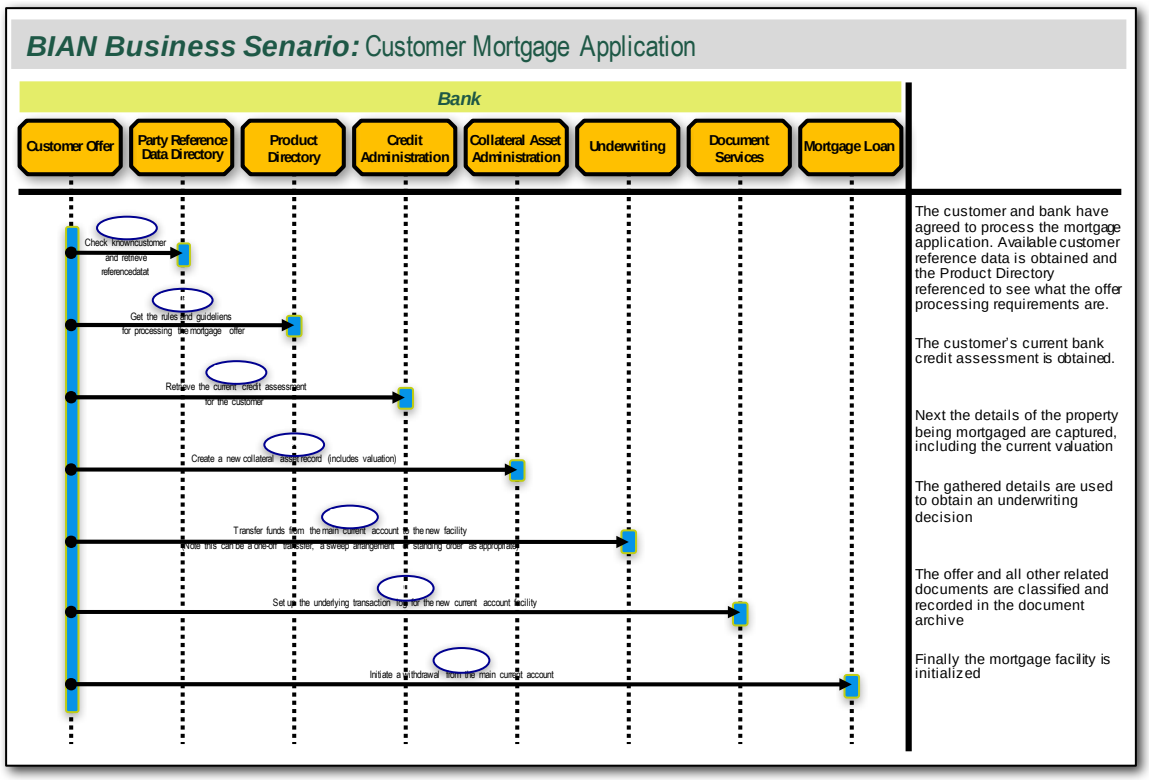


Figure 30 - BIAN Mortgage Application Business Scenario (repeated)

The control record artefact of Customer Offer is a 'customer offer procedure'. Its life cycle covers the offer application process from end-to-end and it contains all business information gathered and created throughout the offer process. Its interactions with the other Service Domains in the scenario and their respective control records are briefly:

- Customer Offer first accesses Party Reference Data Directory to obtain and potentially update customer reference information during the offer process. The control record for Party Reference Data Directory is the party reference data record. Its life cycle covers the duration of the party/customer relationship and is clearly maintained independently of the offer procedure. Pertinent customer information is simply exchanged through a delegated service call

- Customer Offer next references Product Directory to obtain the offer processing requirements for the selected mortgage product. The control record for Product

Directory is the product specification record. Its life cycle covers that of the product and this is also clearly maintained independently of this single offer procedure. Information is again exchanged through services.

- This same type of access pattern can be described: for Credit Administration to review the customer's credit assessment/report: for Collateral Asset Administration to establish or reference a record that is maintained for a collateral item; for Underwriting to obtain an underwriting decision; for document services to access and or capture documents; and finally, Mortgage Loan to initiate the set-up of the mortgage product itself as it starts its own life-cycle in this case.

The examples clarify that matching function and information to the Service Domain is best done by considering the control record and its associated life cycle. Clearly reconciling business function and information with Service Domains is an easier exercise when all of the target Service Domains in play have been identified. This is why developers benefit greatly from having the BIAN business scenarios and wireframes covering their area of interest to reveal all the involved Service Domains.

Externalization is used to ensure business responsibility can be uniquely assigned to/associated with a Service Domain. There are some similarities with the concepts of externalization and the good design of re-usable software utilities. But re-usable software utilities, that can be implemented as a service enabled functions, should not be confused with Service Domains. SW utilities will typically also have clearly bounded functionality and encapsulate their data. A SW utility can also be implemented as a reusable service based software solution component such as a micro service.

The key difference is that though the SW utility functions as an autonomous capability, it does not represent a uniquely assignable business responsibility. By definition there can be many concurrent instances of a SW utility operating completely independently. The utility implementation ensures that the logic is applied consistently and improves software integrity and development productivity but it does not specifically address the operational re-use of a discrete business capability. Not surprisingly a SW Utility will typically be much finer grained than a Service Domain.

Logical Design Extensions

The logical design extensions that solution architects/designers add to the Service Domain conceptual requirements and semantic control record attributes can be made in

any suitable form to suit the development environment and techniques being employed on their project. The general categories of design extension include:

- **Variations** - adding detail that may be specific to support advanced or differentiated behaviors, detail to take account of scale requirements (for larger enterprises), detail to handle geopolitical specific needs
- **Design Options** – selecting between the possible working approaches available such as support for interactive versus off-line processing, or the support for different delivery channels
- **Organizational Arrangements** – handling the particular geographic distribution and different lines of business that make up an enterprise (see below)
- **Non-functional Requirements** – target goals can be defined for the application covering properties such as performance and security.

Most of these extensions can simply captured as expanded requirement definitions associated with the individual Service Domains. In the case of the organizational arrangements however it can be necessary to deploy different versions of the same conceptual Service Domain component.

Organizational Configurations of a Service Domain

When the conceptual Service Domain components are related to a complex organization with different geographic locations of operation and lines of business the functions supported by some Service Domains become distributed as they are necessarily repeated across the organization. There are two dominant patterns for dealing with this distribution:

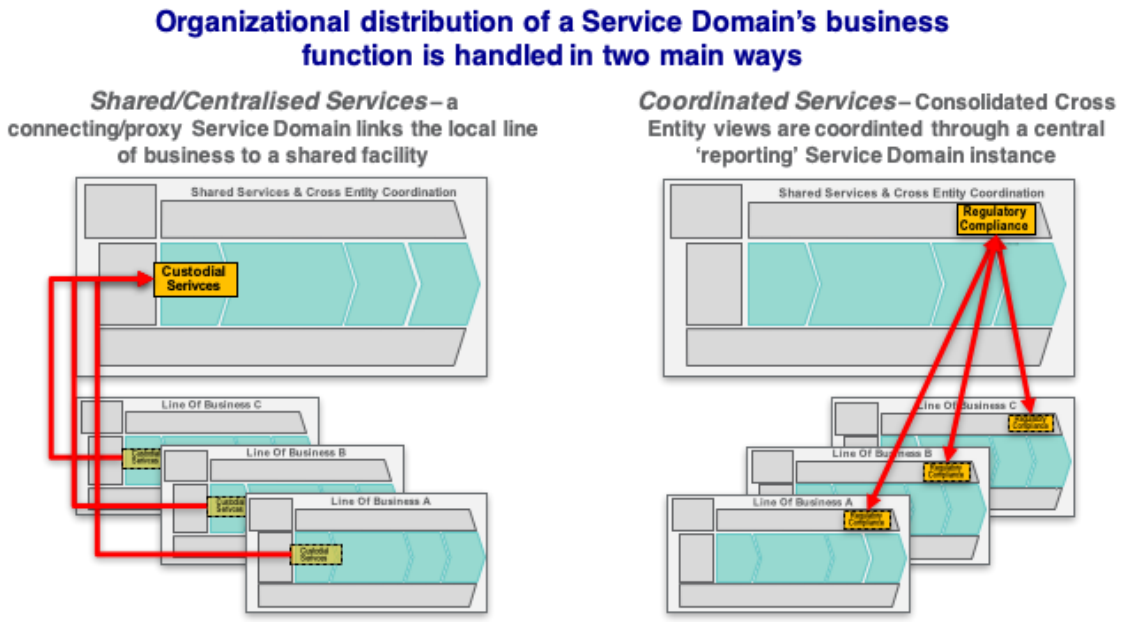


Figure 31 - Two Distribution Options

The two patterns for distribution are either to connect local needs to a centralized implementation or to support local capabilities independently and implement some form of consolidation capability to coordinate between them. The examples in the diagram show when either approach might better suit a specific business situation.

To support organizational configuration a Service Domain may be deployed in four different implementation forms:

- As a local proxy that provides access to a shared centralized service
- As the central service supporting multiple proxies
- As the local fulfillment capability but with reporting obligations to a coordinating 'parent'
- As the central consolidation and coordination 'parent' capability

Application Clusters

As already stated, the Service Domain can be considered a major application module and an application will typically combine several Service Domains. The selection of Service Domain components for inclusion in an Application design needs to balance a number of factors such as functional synergies, technical requirements, performance considerations, legacy application conditions and commercial requirements. The

possible influences are so wide-ranging that it is difficult to be prescriptive as to 'ideal' Service Domain combinations for any specific application. By adopting a component architecture, the ability to assemble any sensible combination is made easier.

When Service Domain components are assembled into a free standing application it is necessary to define different configuration roles they might play as outlined in the example:

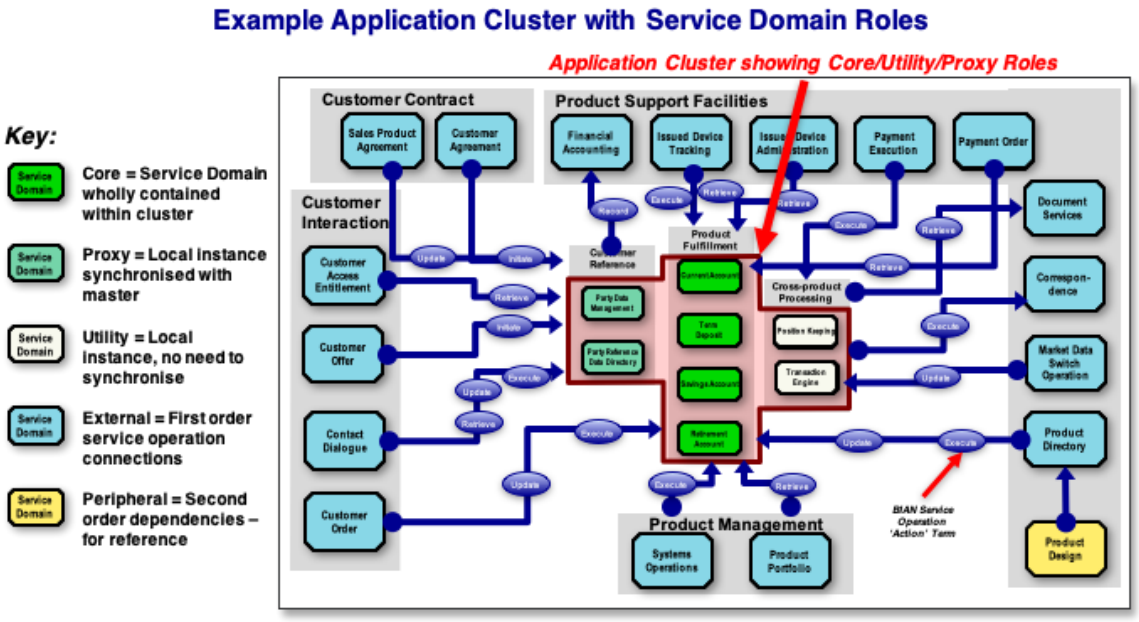


Figure 32 - Application Cluster

From the diagram five distinct roles can be seen for any involved Service Domain:

- **Core** – the Service Domain instance running in the application is the single, master version for the enterprise. It is the only physical instance and the sole source for its services and information
- **Proxy** – the Service Domain running in the application supports all local requirements but is connected to an external 'master' Service Domain (that will be running as a 'Core' Service Domain in some other application
- **Utility** – is a local/proxy implementation of a Service Domain where due to its specific business role, it can operate with no or limited need to connect and synchronize with a master/core version

- **External** – records that there is a direct connection from the application cluster to the Service Domain to either offer and/or subscribe to services. These define the main external application interfaces
- **Peripheral** – Sometimes it helps to include additional Service Domains in the cluster diagram that have some indirect involvement (through an External Service Domain) simply to clarify limitations in the external application boundary

The logical design for the Service Domain needs to be developed in alignment with its type of role as outlined in the application cluster. This can involve designing some level of background service based coordination with external applications to deal with the different physical configurations.

4.2.3 Physical Specifications

The physical specifications cover the actual code and data specifications used to implement the Service Domain functionality. The BIAN standard as well as being implementation agnostic, does not assume or impose any physical properties for the Service Domains. That said when applied in a component/container type deployment there are some operational properties that can determine the type of software architecture and utilities that might be most suitable.

A checklist of some software approaches and utilities to consider include:

- ***Message queues and events*** – service exchanges and service triggering, including sequencing, security and resilience features will apply to all types of Service Domain

- ***(Finite) State machines*** – can be applied to govern the control record lifecycle and to its sub-partitions as defined by the behavior qualifier type as necessary

- ***Event driven processing*** – in partner with state machine designs, there is wide potential to leverage event driven design. This can apply to specific attributes and their associated rules/policies or to the states and transition patterns of the control records (and their behavior qualifier defined sub-partitions)

- ***Workflow management*** – will have broad application for most types of Service Domains. In some cases, it will be appropriate to ‘nest’ workflows perhaps aligned to the control record breakdown by behavior qualifier type

- ***Rules engine*** – as with workflow management – rules engines are likely to have wide application

- ***Data management utilities*** – particularly as Service Domains govern their own autonomous data repository, a broad range of data management facilities will be required. Advanced data management features are likely to have specific relevance in highly distributed environments (for replication and resilience)

- ***Analysis and reporting facilities*** – General analysis and reporting will be widely applied

- ***Command & Control*** – As each Service Domain can act as its own operational unit there is the possibility to develop Service Domain aligned standard tracking and reporting facilities to assist with the implementation of command and control structures between Service Domains

Some utilities may sensibly apply to all Service Domain partitions when implemented as containers in a SOA. Some utilities may be better suited to Service Domains with specific functional patterns of behavior. For example, a 'Process' Service Domain is likely to make significant use of workflow management utilities. The table below provides an indication of how different SW utilities might be particularly well suited to a Service Domain based on its Functional Pattern:

Mapping Functional Patterns to general SW Utilities (indicative)

Functional Pattern	Description	Message Queues	State Machines	Event Processing	Workflow Management	Rules Engines	Data Management	Analysis & Reporting	Command & Control
DIRECT	Define the policies, goals & objectives and strategies for an organizational entity or unit	●	◔	◔	◑	○	◔	●	●
MANAGE	Oversee the working of a business unit, assign work, manage against a plan and troubleshoot issues.	●	◔	◔	◑	◔	◔	●	●
ADMINISTER	Handle and assign the day to day activities, capture time worked, costs and income for an operational unit.	●	◔	◔	◑	◔	◔	●	●
DESIGN	Create and maintain a design for a procedure, product/service model or other such entity.	●	○	○	●	○	●	●	●
DEVELOP	To build or enhance something, typically an IT production system. Includes development, assessment and deployment	●	○	○	●	◔	●	●	●
PROCESS	Complete work tasks following a procedue in support of general office activities and product and service delivery functions.	●	◔	◔	●	◑	◑	●	●
OPERATE	Operate equipment and/or a largely automated facility.	●	◔	◔	●	◑	◔	●	●
MAINTAIN	Provide a maintenance service and repair devices/equipment as necessary.	●	◔	◔	●	◔	◑	●	●
FULFILL	Fulfill any scheduled and ad-hoc obligations under a service arrangement, most typically for a financial product or facility.	●	◑	●	●	●	●	●	●
TRANSACT	Execute a well bounded financial transaction/task, typically involving largely automated/structured fulfillment processing.	●	●	●	●	●	●	●	●
ADVISE	Provide specialist advice and/or support as an ongoing service or for a specific task/event	●	◔	◔	●	◔	◑	●	●
MONITOR	To monitor and define the state/rating of some entity.	●	●	●	○	◔	◔	●	●
TRACK	Maintain a log of transactions or activity, typically a financial account/journal or a log of activity to support behavioral analysis.	●	◑	◑	◔	◑	●	●	●
CATALOG	Capture and maintain reference information about some type of entity.	●	◔	◔	◔	◑	◑	●	●
ENROLL	Maintain a membership for some group or related collection of parties.	●	○	○	●	◔	◑	●	●
AGREE TERMS	Maintain the terms and conditions that apply to a commercial relationship.	●	○	◔	●	◔	◑	●	●
ASSESS	To test or assess an entity, possibly against some formal qualification or certification requirement.	●	○	○	◔	●	●	●	●
ANALYSE	To analyse the performance or behavior of some on-going activity or entity.	●	○	○	◔	●	●	●	●
ALLOCATE	Maintain an inventory or holding of some resource and make assignments/allocations as requested.	◔	◔	◔	◑	◑	◑	◔	◔

Figure 33 - Functional Patterns Mapped to SW Techniques & Utilities

In summary, the BIAN standard provides conceptual component designs and provides some guidance as to how these requirements can be interpreted in development where more comprehensive logical designs and physical specifications are required. The underlying assumption is that to remain implementation agnostic and support a canonical definition, the BIAN definitions must be focused at the conceptual level.

The intended scope of BIAN's coverage is indicated:

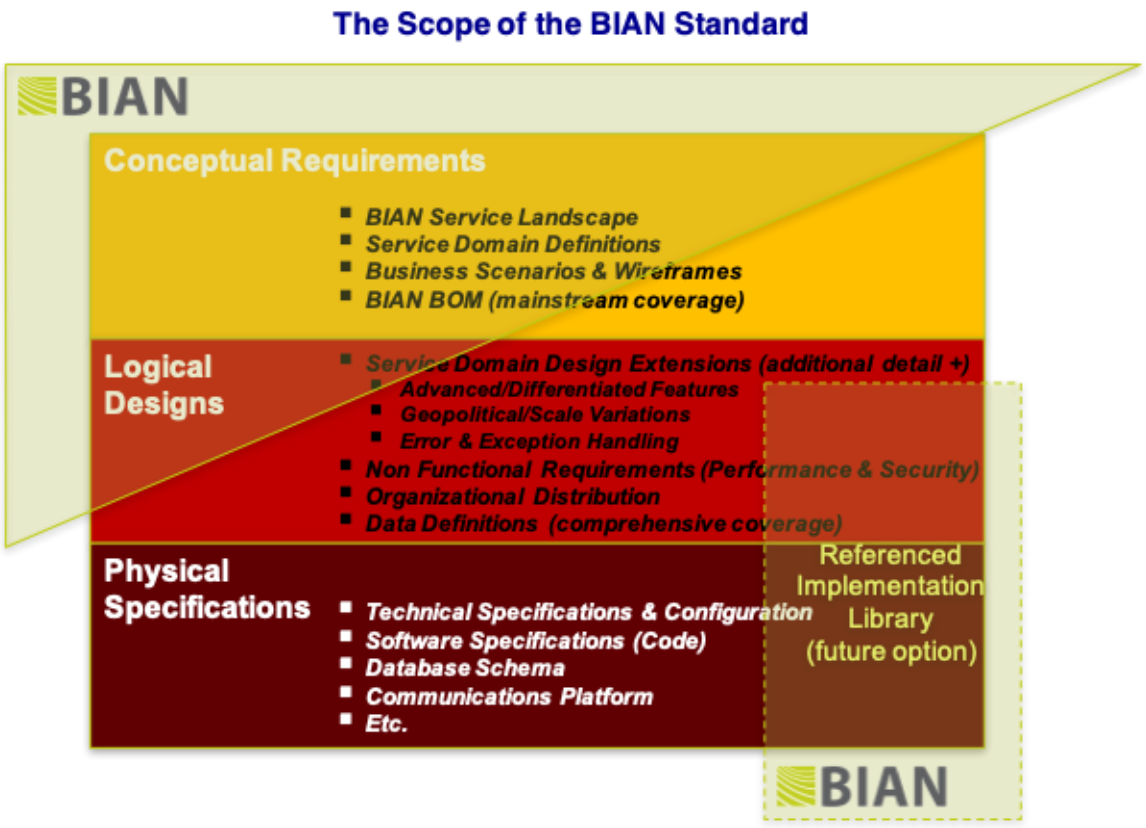


Figure 34 - Scope of BIAN Against the Conceptual/Logical & Physical Layers

4.3 Implementation Approaches

The implementation approaches detailed here outline insights that BIAN members have identified for leveraging the Service Domain component partitions in physical implementation. Some of these techniques have been adapted from techniques used before in related situations such as using BIAN Service Domains and service operations as a basis for organizing enterprise service bus (ESB) integration.

This is a brief initial list that will hopefully be extended as new techniques and insights emerge from practical experience. The list is split between approaches suited specifically to legacy renewal and those that can apply to both legacy and green field development.

Legacy Wrapping Specific approaches

- Externalization applied to legacy application modules
- Reconciling master/slave information governance
- Wrapping & service enablement
- Migration strategies – the parallel core configuration

General Approaches (suited to both legacy wrapping & greenfield development)

- Shared platform to eliminate service exchanges
- Shared platform to support consolidated cross Service Domain reporting
- BIAN Type 1,2 & 3 external access governance patterns

4.3.1 Legacy Wrapping Approaches

Legacy wrapping is an approach that protects the investment in existing systems by seeking to enable them to operate in a service based architecture where appropriate. Wrapping 'compartmentalizes' legacy systems aligning to component boundaries with the wrapping technology providing a mechanism to mitigate shortfalls in the legacy application. Many legacy systems suffer from fundamental architectural limitations such as inflexible/monolithic structures and operating in batch mode. But they also often contain extensive and proven business functionality that would be prohibitively expensive to re-create.

As already noted, systems that align to a component architecture tend to be more resilient and flexible as they more readily support different processing flows and are more adaptive to changes. Once legacy systems have been wrapped and shortfalls mitigated their shelf-life may be extended considerably. Depending on the extent to which the wrapping approach masks the architectural limitations of legacy systems it is possible that key areas of the application portfolio can be repurposed and retained for a significant time. Particularly for legacy application that already operate in real-time.

The main focus for legacy wrapping is likely to be repurposing high throughput back office transaction processing systems. The component blueprint is particularly useful for a broad range of host wrapping and migration techniques. This is because the component partitions defined in the BIAN standard are highly stable over time.

As described in the *BIAN How to Guide – Applying the Standard*, an enterprise can develop an organizational blueprint built using Service Domains as the building blocks. Once assembled, as long as the business does not change its lines of business or locations of operation the enterprise blueprint will not change.

An enterprise blueprint is similar to the more narrowly focused wireframe diagram (as described in Section 3.4) that covers a specific business area. Both provide a stable framework for migration exercises. Current applications can be mapped to the Service Domain components based on their functional content. The Service Domain component service exchanges shown in the wireframe view can also be used to define the major application interfaces (APIs).

Once a legacy application is mapped to the component blueprint it is possible to undertake a progressive migration, making incremental changes targeting the more severe constraints or the greatest opportunities to fund the transition. Clearly planning an incremental migration of a complex legacy system is a major undertaking. But as many hosts systems reach obsolescence this migration is no longer optional and an approach that allows this to be done incrementally against a stable long-term operational blueprint has obvious significant benefits.

Externalization applied to legacy application modules

Externalization is the BIAN technique used to define the functional scope of a Service Domain component – determining what it handles directly and what it delegates to other Service Domains. The approach as applied to define Service Domains has already been described in Section 4.2.1. of this guide. It is a simple adaptation of this technique that can be used to apply the same control record based evaluation to the functional scope of legacy applications in order to map to the corresponding Service Domains.

The control record based mapping used in the externalization technique can be combined with the Application Cluster perspective described later in the same Section 4.2.1. to develop a target component perspective for the legacy application. The approach to develop the target state component model for the wrapped legacy application includes these main steps:

1. Working through the functional scope of the legacy application and referencing the Service Domains and their associated control record specifications, identify mapped Service Domains
2. The layout of an Application Cluster diagram is then initiated to represent the target state/boundary for the wrapped legacy application.
3. The application cluster diagram is populated as key decisions are made as to whether mapped Service Domain component functionality should remain within the application cluster or be externalized.
4. For Service Domain mapped functionality that is to remain within the wrapped application a further decision must be made as to whether it is to represent the core or a proxy instance of this functionality for the enterprise.
5. For all other mapped functionality, the associated Service Domain capabilities should eventually be sourced externally from alternate applications.

6. Finally, for key external interfaces to the legacy application, the existing interfaced system should also be mapped to Service Domains to ensure there is no redundancy/overlap with the target application Service Domain component make-up

Reconciling master/slave information governance

The Service Domain component mapping to the legacy application just described reveals those components that are included within and those that are external and that need to be interfaced with the wrapped application. The Service Domain control records can then be used to determine the information governance responsibilities with specific attention to the included Service Domains.

As described in Section 4.1.2 the design of the Service Domain components results in the unique mapping of all business information to individual components. The key information governed by a Service Domain component is catalogued in the semantic attribute definitions of its control record. At a fairly high level of detail, the control records for the collection of included Service Domains define the primary governed information for the application.

Note there may be global referenced data and other Service Domain related data but these aspects are not considered for simplicity at this stage. See the complete description of the Service Domain Information profile earlier in this guide.

The governed information inventory can then be related to the current legacy application database/information and categorized as follows:

- It represents Master data that is governed by the application – and so must provide external access as necessary
- It represents Proxy Master data that is governed in another instance of the Service Domain in some other application. Reconciliation services need to be established to synchronize with this external source
- Is a local copy of externally governed information – i.e. should be retrieved through an external interface/service call and the values interpreted and applied to the internal business information model as appropriate

Though high level, the control records provide an inventory of the governed information matched to the contained Service Domain components of the wrapped legacy application. This helps define the future state information architecture for the wrapped application.

Wrapping & service enablement

The two prior sub-sections focus on functional and information aspects of component based legacy repurposing. Wrapping and service enablement augments the component mapped application with mitigating functionality and if appropriate implements a service based communications capability to support external interfaces.

The mitigation logic provides interim capabilities that address shortfalls in the legacy application. These might be resolved as the legacy software is reengineered within component aligned containers, enhanced in place or replaced completely. The types of mitigating logic that can be built into the container architecture include:

- **Functional extensions** – adding 'front-end' functionality and supporting operating requirements that can't be easily built into the legacy codebase
- **Synchronization** – capabilities to handle the master/slave data synchronization requirements between systems
- **Proxy capabilities** – support temporary functions that will eventually be provided by alternate/external service providers
- **Session optimization and data caching** – the wrapper may streamline access management and can also include logic that performs advanced probabilistic data look-up and caching to reduce host access costs and latency

In addition to the wrapping logic, service enablement capabilities involve establishing support for queue and event driven service handling for offered and delegated service interfaces. These can progressively replace exiting point to point interfaces. The service management capabilities handle information extraction and import, message assembly and the protocol/choreography of the service exchanges as appropriate. This may include tracking and matching responses over time and other scheduling concerns

The service enablement extensions may also provide more general facilities such as service directory support, service subscription and service level agreements, access control/security, performance assurance and reporting functions.

Migration strategies – the parallel core configuration

There are numerous established approaches to legacy migration. The key advantage of a component blueprint in any migration as already noted is that it provides a stable view of the 'to be' state that incremental developments can build towards.

When the component blueprint is combined with a detailed underlying data specification a powerful migration option can be considered – the ‘parallel core’ configuration. In the parallel core configuration, a comprehensive data model that is defined to the level of a physical data specifications is applied to the target component design. The same data model must then also be mapped to the legacy data structures.

Once this detailed data mapping work is complete the migration involves building out a parallel component capability in parallel with any componentization/wrapping of the legacy application. As the data is mapped, the mirrored capabilities in the parallel core match the legacy application (this may not be in real-time). Initially the parallel core may simply provide background capabilities such as off-line analysis and reporting. But the core can be designed to be capable of eventually replacing its matched legacy component.

In time as critical mass is captured in the parallel core the legacy host components can be retired from production. When all component capabilities have been migrated the host application can be decommissioned.

The parallel core migration is critically dependent on the availability of the physical data specifications. Some examples of an industry standard data specifications are available. BIAN is currently undertaking feasibility pilot project with such a model as provided by Ariadne Inc. (their ACTUS open financial instrument cash-flow contract standard).

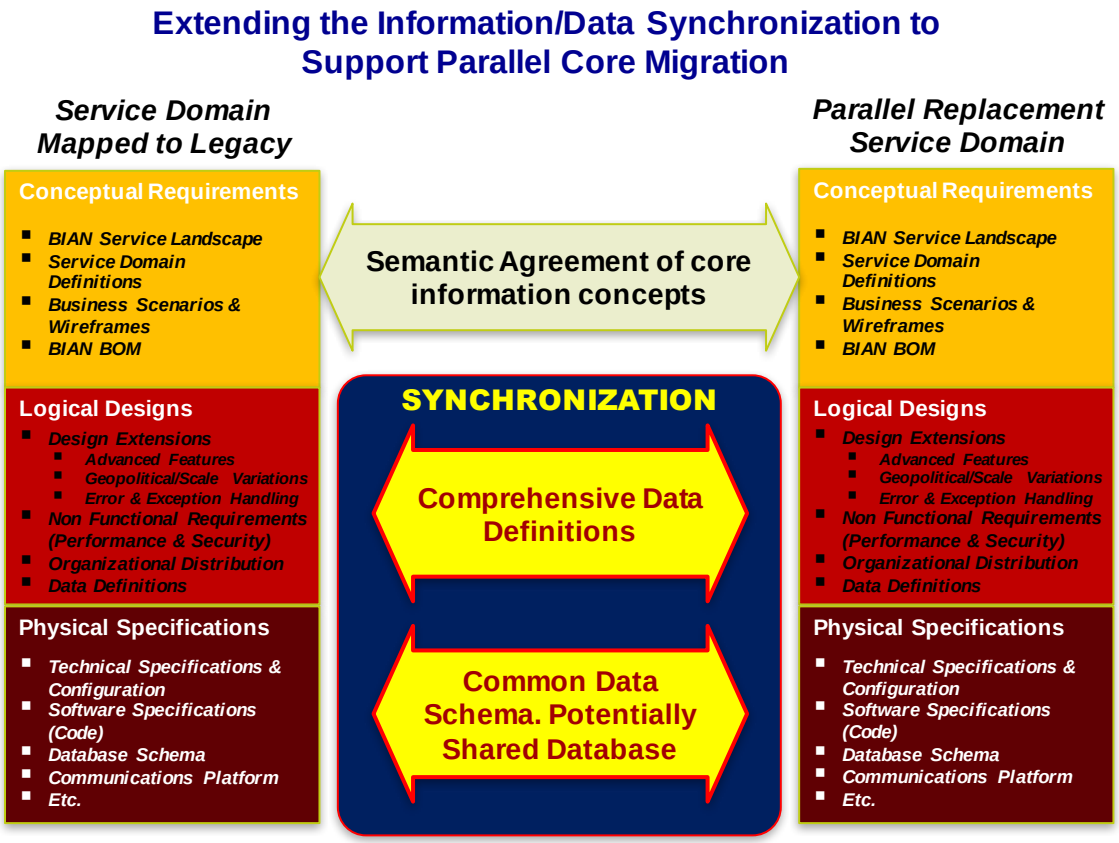


Figure 35 - Parallel Core Service Domain Migration

4.3.2 General Approaches (for legacy wrapping & greenfield development)

The current list of general approaches includes:

- Shared platform to eliminate service exchanges
- Shared platform to support consolidated cross Service Domain reporting
- BIAN Type 1,2 & 3 external access governance patterns

In addition to these targeted techniques developers should reference the checklist of advanced implementation concepts outlined for an event driven component architecture earlier in this guide (Section 3.6).

Shared Platform to Eliminate Service Exchanges

The component architecture defines discrete and re-usable business functions that can be shared by means of standard service interfaces. Implicit in this model is that the connections are typically service enabled in implementation. However, and as noted in the many earlier discussions of back office process oriented designs, for performance reasons some exchanges may need to be implemented as dedicated point to point connections (eliminating the latency and additional layers of processing typically required in a service exchange).

Another physical configuration can be considered when even higher performance is required. In this approach as with the parallel core approach already described, the Service Domains need to be aligned at the physical data specification level. In this approach the two Service Domain's maintain their own logical information perspectives, but the exchanged information attributes are mapped to common physical storage. Data management and access controls are required to manage concurrent access but updates made by one Service Domain are instantly visible to the other. The information is logically exchanged, but no actual physical data transfer takes place.

This configuration is indicated in the diagram:

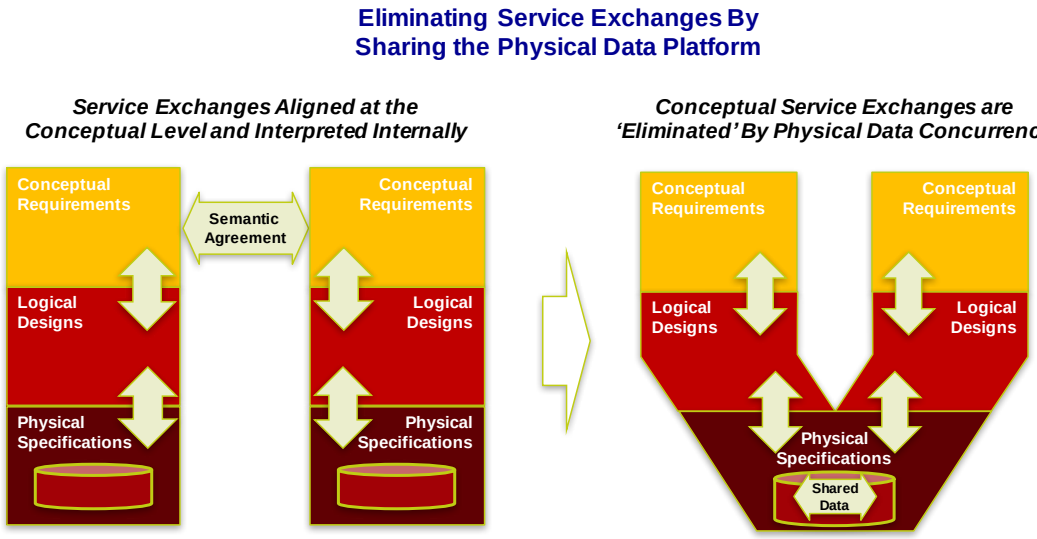


Figure 36 - Eliminating Service Exchanges

Shared Platform to Support Consolidated Cross-Service Domain Reporting

The shared platform approach can be taken further to support high-performance reporting requirements. In this situation one Service Domain is responsible for the consolidation, analysis and reporting on information obtained from potentially multiple source Service Domains.

As with the shared platform between two Service Domains it is necessary for all involved Service Domains to agree the physical data specifications for all exchanged data. The coordinated access to update the shared physical data may require more sophisticated access management to ensure the integrity of the single physical consolidated 'position'. But this physical configuration can support powerful real-time information tracking and reporting needs for sensitive transactional information.

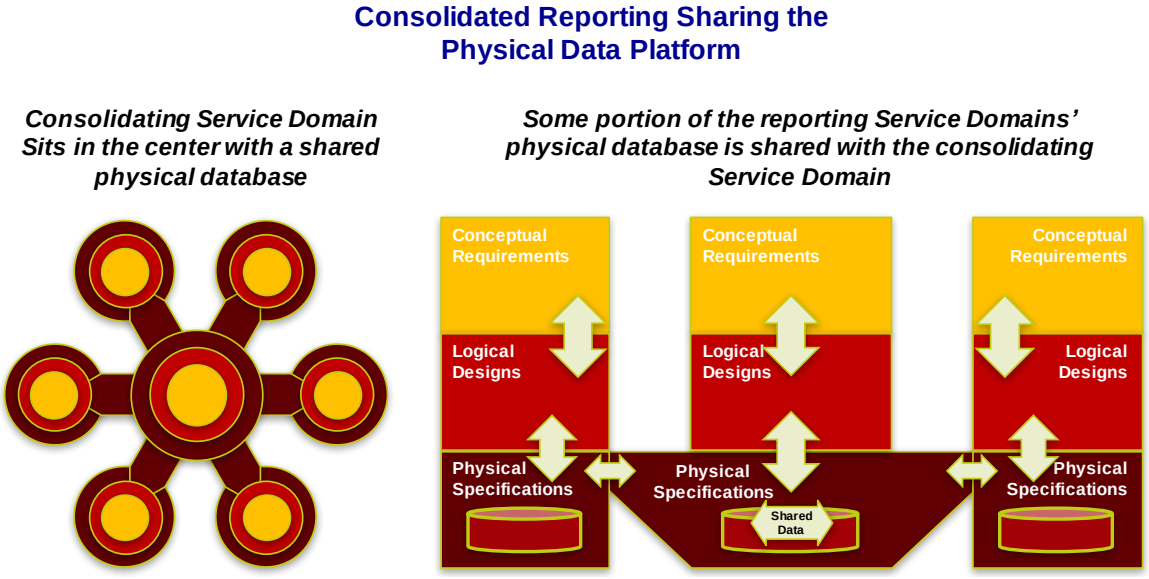


Figure 37 - Shared Platform for Consolidated Reporting

BIAN Type 1,2 & 3 external access governance patterns

Because the Service Domains collectively cover all banking activities and are each discrete/non-overlapping, their service operations can be used to define a comprehensive industry standard services directory. Some service exchanges will occur solely within the bank, between its application mapped Service Domains – these are referred to as application to application (A2A) APIs. Other service operations may be provided to parties (customers or third parties) external to the bank – these are referred

to as bank to business and bank to customer (B2B/B2C) APIs. As the distinction between A2A and B2B/C can vary by enterprise and given that to fully support any business event it is necessary to address all involved exchanges (both any A2A and B2B/C interactions) BIAN defines all Service Domain operations to the same level regardless.

As the BIAN Service Domain service operations are defined at the conceptual level they may not always map to an individual service exchange with a single request and response message pair (i.e. an “endpoint”) in physical implementation. The exchange may need to be further broken down to multiple constituent endpoints fully represent the choreography and allowed options that make up the interaction. But the BIAN service operation in this case does at least provide a unique and discrete classification of the purpose for the exchange.

For service operations that support external access three distinct patterns have emerged as outlined in the table:

Three Distinct Types of External Access

	Type 1. Direct to Core	Type 2. Wrapped Host	Type 3. Component Architecture
Definition	The API routes direct to the core system providing the service. Intermediate channel based access control and ‘buffering’ is required	Integrating service middleware – a service bus – ‘wraps’ the host systems. The service bus can offer various host access mitigation capabilities/enhancements	The host services are implemented as loose coupled micro-services with complex interactions supported by sophisticated connective middleware
API Service Description	Read only or simple ‘atomic’ update transactions supported by a single host system. The solution is likely to be host application specific	Enhanced ‘simple access’ services aligned to established standards. Wrapping may enhance service capabilities and some hosts may support more complex exchanges	Support for flexible and complex interactions involving multiple business activities and processing/decision chains
Examples	◆ 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 technologies/architectures
Pros & Cons	Pro - Easy to implement Pro - Re-uses interfaces Con - Can’t do multi-phase Con - Not handle automation	Pro - Reuses legacy capabilities Pro - Mitigates some shortfalls Con - Limited for multi-phase Con - Limited for access controls	Pro - Supports sophisticated interactions Pro - Supports flexible models Con - Complex to implement Con - Operational overheads

Figure 38 - Three Types of Access

The three types of access are briefly:

1. **Direct to Core** – the external access is governed by a gateway that implements basic customer authentication. The gateway then connects directly to the host system. In many situations re-using an existing external interface that might have been implemented to support web based or contact center servicing

2. **Wrapped Host** – in addition to the access gateway, the wrapped host approach includes a front end capability that can address shortfalls in the host systems. This can support host migration and repurposing efforts. It includes coordinating access with multiple systems for more complex transaction, resolving master slave data conflicts, host access session optimization, information advanced look-up and caching and supporting functional extensions

3. **Component Architecture** – involves a comprehensive set of controls to manage external access to allow direct connection to the internal capabilities of the bank. A wireframe of the external access platform is shown below

The main elements of the three type of approach are shown in the diagram below. Note that most banks will probably need to support some combination of all three types of access approach.

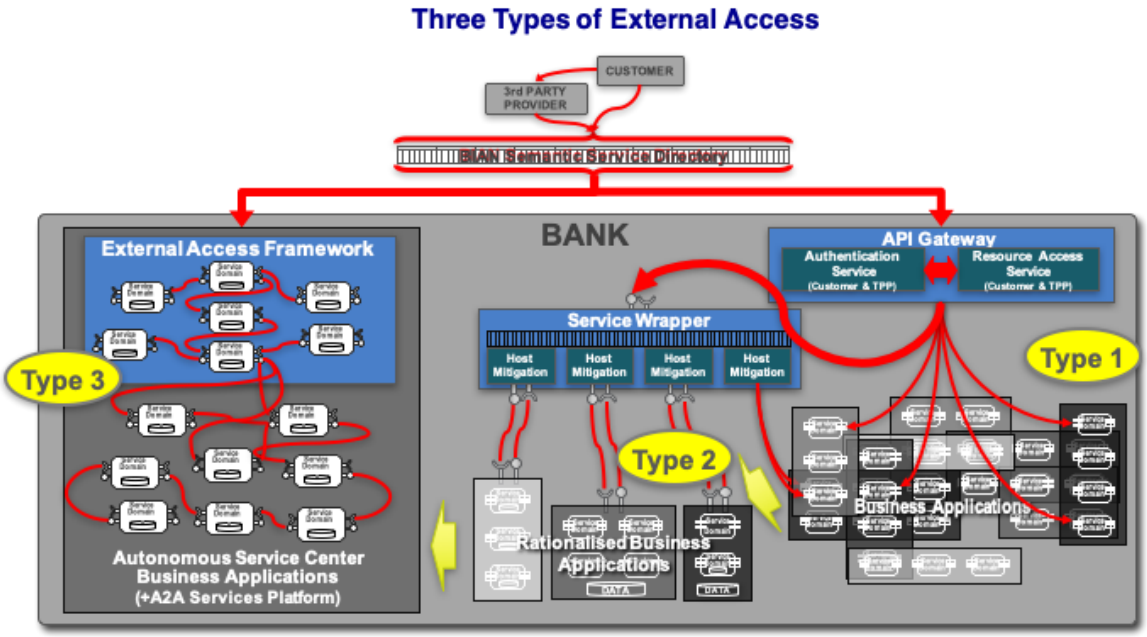


Figure 39 - Three Types of Access Schema

The Type 3 access approach is particularly significant as banks consider their approach to open banking. When the Type 1 & 2 approaches are implemented to support third party access the connection is typically made directly to the back office transaction processing systems. This in effect eliminates the bank from participating in any 'front office' dialogue with the customer and third party solution provider.

The Type 3 access approach also supports third party access but links the customer contact to a front office Service Domain – Session Dialogue. This can then act as a gateway that structures access to the bank. If appropriate it can access the back office transaction systems directly (as in Type 1 & 2 approaches). But it can also support a customer interaction where a wide array of front office capabilities can be worked into the customer interaction. The Session Dialogue Service Domain can optionally leverage a second front office Service Domain: Servicing Order that implements more complex structured workflows to orchestrate different customer interactions.

The distinction between Type 1 and Type 2 & 3 access patterns is summarized in the following diagram:

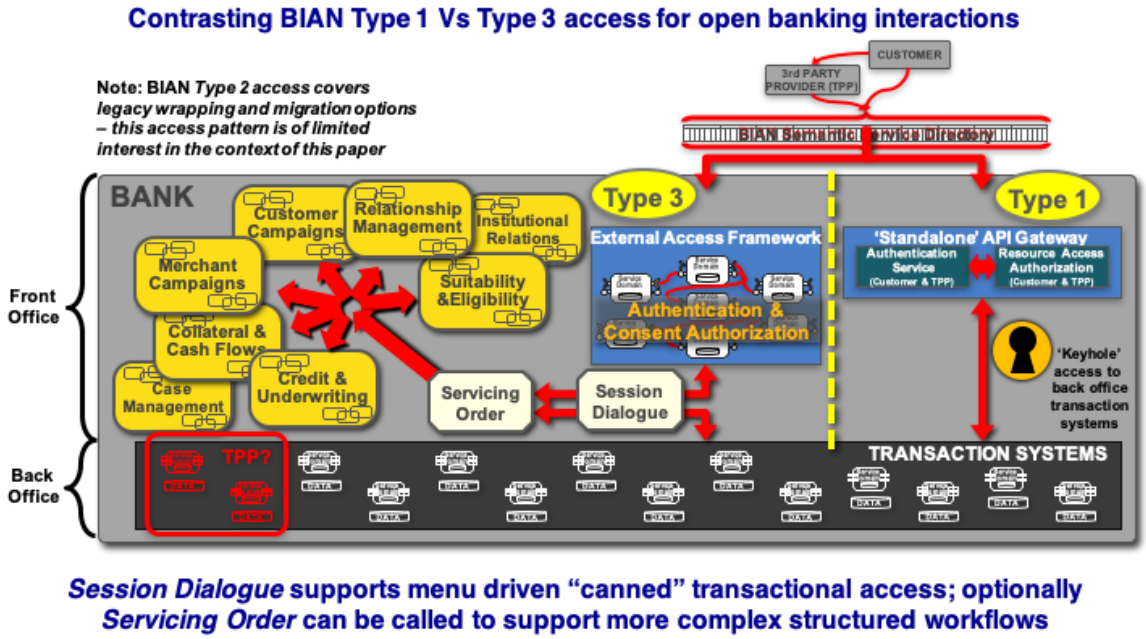


Figure 40 - Contrasting Type 3 and Type 1 & 2 Access

Finally, the main elements of the Type 3 External Access Framework can be seen in the following wireframe. The wireframe reveals the range of capabilities needed to properly control the access of customers and external third parties to the internal workings of the

bank. As bank's explore new business models, including collaboration with FinTechs and the adoption of open banking approaches, support for this kind of external coordination will be crucial.

The BIAN External Access Framework is a draft specific application of the BIAN standard that is being applied to a proof of concept initiative at the time of publication. The results of this exercise and a more complete specification of the framework is documented elsewhere.

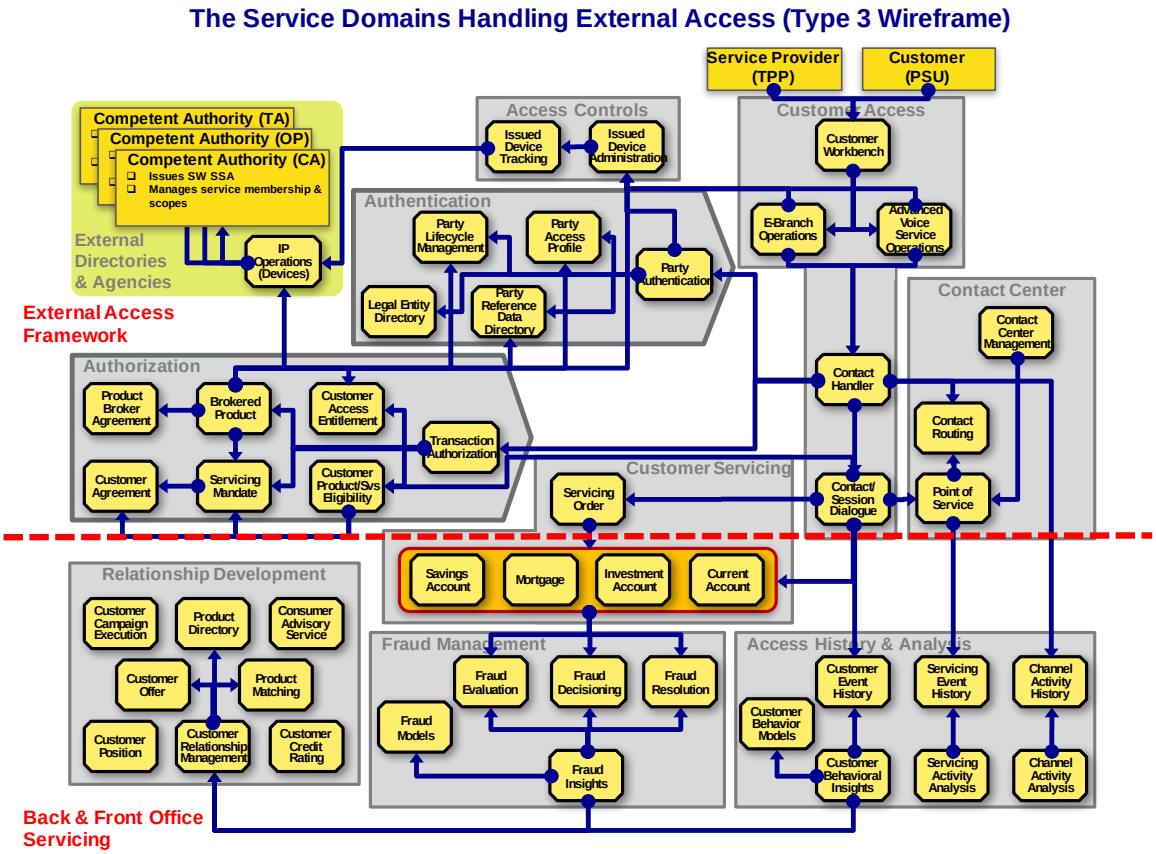


Figure 41 - External Access Framework Wireframe

Attachments

A - Action Terms as they Relate to Functional Patterns & Control Records
B - Right-sizing a Service Domain

ATTACHMENT – A – Action Terms Related to Functional Patterns

Action Terms as they Relate to Functional Patterns & Control Records

In some cases, it can appear that the same action term results in different behaviors when applied to different Service Domains. This is because the Service Domains' can have very different underlying operational characteristics. To fully understand how the action term is actually being consistently applied it is sometimes necessary to consider the Service Domain's functional pattern/control record explicitly.

The is the case with the response to the action terms initiate and execute when applied to Service Domains with a *fulfillment* functional pattern compared to Service Domains with a process functional pattern. To clarify we will review these actions terms applied to two different Service Domains: Current Account with its fulfillment functional pattern and Customer Billing with its process functional pattern. In each case we also consider how a sample of their underlying behavior qualifiers are accessed.

The Current Account Service Domain with the 'fulfilment' functional pattern has control record instances: *current account fulfilment arrangements*. The behavior qualifier type for a fulfilment arrangement (i.e, how the *current account fulfillment arrangement* record is broken into parts) is *features*, in this case representing the different product features that make up the current account facility. For this explanation we will refer to two behavior qualifiers/product features – *interest* (handles the array of interest rates applicable to the current account facility) and *payments* (handles the set-up and execution of different types of payment made from the account, including regular payments, standing orders, direct debits and bill pay).

The Customer Billing Service Domain with the 'process' functional pattern has control record instances: *customer billing procedures*. The behavior qualifier type for a process (i.e. how the customer billing procedure record is broken into parts) is *work steps*, in this case the series of steps in handling a customer billing cycle. For this explanation we will refer to two behavior qualifiers/procedure work steps – invoicing (handling the generation of the customer invoice) and *tracking and reminders* (handling the tracking and issuance of reminders).

The action terms *initiate* and *execute* have very different general purposes as already described. Below we describe how they both act on the two different Service Domain types to clarify how they do in actuality act consistently on their control records (or their constituent behavior qualifiers). But as can be seen the requests result in rather different operational behaviors due to the differing functional patterns:

> ***Initiate*** – results in the creation and initialization of a new control record instance or a contained behavior qualifier instance for an existing control record. This

action term is used for both the Current Account and the Customer Billing Service Domain.

For Current Account the initiate action term results in the following response at the control record and behavior qualifier levels:

> “initiate current account fulfillment arrangement” will result in a new current account facility being established and initialized as appropriate
>
> “initiate current account fulfillment arrangement | interest” will result in specific interest handling features being established for the account. This would normally be done as an internally orchestrated product fulfillment set-up function so an external service call might not always be required/supported
>
> “initiate current account fulfillment arrangement | payments” will result in the set-up of a payments capability associated with the account. This includes regular scheduled payments such as a standing order. In the case of one-off/ad-hoc payments this call only establishes/configures the capability to handle payments – importantly it does not handle the transaction itself (see ‘execute’ later)

For Customer Billing the initiate action term results in the following response at the control record and behavior qualifier levels:

> “initiate customer billing procedure” will trigger the customer billing process, in this case the end to end processing of a customer bill. As the Service Domain’s process logic may orchestrate all end to end actions, this might be the only required external service call
>
> “initiate customer billing procedure | invoicing” would trigger the generation of the invoice that is then sent to the customer. As this work step follows on automatically from the initiation of the overall process, this more specific service is unlikely to be required/supported as just noted
>
> “initiate customer billing procedure | tracking and reminders” would trigger the generation of a reminder missive if the payment is overdue. This action could also be internally generated to a schedule or it is possible that an external service would be provided to allow other parties to trigger billing reminders

Execute – is an action that acts on an active control record instance or one of its subordinate behavior qualifier instances as necessary. The execute action will invoke some automated task that is then applied to the instance.

For Current Account the execute action can result in the following responses:

> "execute current account fulfillment arrangement" will trigger an automated action that applies to the overall account such as perhaps purging old account records (it is hard to define a particularly good example as most interesting product functions are covered by the underlying behavior qualifier product features)
>
> "execute current account fulfillment arrangement | interest" will trigger an automated task associated with the application of interest to the account – such as applying an amended rate to some specific aspect of the account
>
> "execute current account fulfillment arrangement | payment" will trigger a payment transaction against some pre-configured payment feature. This could be making an ad-hoc payment from the account, or an instruction to override a scheduled standing order payment for example.

For Customer Billing the execute action can result in the following responses:

> "execute customer billing procedure" will trigger an automated action against an active billing procedure, for example an instruction to reset the billing process
>
> "execute customer billing procedure | invoicing" will trigger an automated action specific to an already active invoicing work step of the billing process such as generating a duplicate/repeat invoice
>
> "execute customer billing procedure | tracking and reminders" will trigger an automated action against an already active reminder work step such as redirecting the scheduled reminder missives

From the example it can be seen that for Service Domains that handle an on-going fulfillment facility such as Current Account the initiate action is needed to set-up some feature and then the execute action is used to trigger related transactional events. For Service Domains that are more process oriented such as Customer Billing the initiate action typically triggers a chain of processing activities and the execute action is only used (comparatively rarely) to intervene in these active processing activities when necessary. The response to the initiate and execute action terms is consistent in each

case, but due to the different operating profiles, the resulting Service Domain responses are quite different.

It is worth noting that the BIAN control records and behavior qualifier types have been specifically designed to ensure the meaning of an action term is consistent whether it is applied to the overall control record instance or any of its constituent parts as defined by the behavior qualifiers (or sub-qualifiers). As described earlier this is because the behavior qualifier continues to apply the functional pattern behavior but to some sub-set aspect of the Service Domain's function.

ATTACHMENT – B – Right-sizing a Service Domain

Right-sizing a Service Domain

The technique used to define the correct scope for a Service Domain is quite complex, iterative and involves several overlapping considerations. As implementers use established Service Domains and should not have to define new Service Domains it is fortunately not necessary to learn this particular technique. It is explained in the BIAN architectural guides but is briefly outlined here for general information only.

As every Service Domain applies a single functional pattern of behavior for the complete life cycle the one variable that determines how a Service Domain is 'right-sized' is the selection of the asset type that it acts upon. In this context an asset refers to anything that the bank owns or has some control over. An asset can be something tangible such as buildings or technology or something less tangible such as a customer relationship or market knowledge. The capacity to perform some kind of function is also considered as an asset such as a call center that provides the capacity to service customers and the production facilities that provide the capacity to deliver current account services.

In order to isolate banking asset types BIAN has defined a mutually exclusive, collectively exhaustive (MECE) asset type classification hierarchy. Asset types are progressively broken down into sub-types up to the precise point where they retain unique business meaning/context. Below this level the finer grained asset types become more utility in nature.

For example, consider the asset representing the overall capacity a bank has to handle interactions with different parties. This asset/capability can be broken down to sub-types that might address interactions with different types of party (e.g. interactions with employees, business partners and customers). At some point, say when we attempt to break down the capacity to handle the interactions with customers further we define finer grained activities such as the capacity to hold meetings, develop performance plans, troubleshoot issues, etc. These actions are no longer uniquely assignable to a specific organizational role (in this case customer relationship management) but are more utility in nature as they can be performed in many different parts of the organization.

Asset types defined at the level just above the point where they are commodity in nature, when acted upon by a single functional pattern define an elemental business function partition. So in this example, applying the 'management' functional pattern, we could define a Customer Relationship Management Service Domain as it has unique business context that can be clearly assigned within the organization. But a lower level "Party Meeting Management" Service Domain could be assigned to many different responsible organizational areas as a utility function/activity and so fails the design requirements for a Service Domain.