Government of Sultanate of Oman

Information Technology Authority

Oman eGovernment Architecture Framework (OeGAF)

Solution Reference Model (SRM)

Revision History

Version	Date of Revision	Prepared / Updated By	Reviewed By	Reason for Change	Affected Sections
1.0	30 Mar 10	IDA International	OeGAF Steering Committee and Core Team	-	-
1.4	28-Dec-2013	ITA	OeGAF Core Team	Removed obsoleted information and added new information	-
1.5	31-Jan-2014	ITA	OeGAF Core Team	Move 2 domains from TRM into SRM	All
1.6	10-Feb-2014	Project Manager	OeGAF Core Team	Added architecture vision for SRM; removed Application Portfolio; added list of figures & tables; removed technical standards into separate sections	All

Table of Contents

List of Figures and Tables

1 Overview

1.1 Document Purpose

Government services to the citizens, residents and commercial establishments can be improved through technology standardisation and service integration amongst the Oman government agencies.

The purpose of the SRM is to document four architecture elements, namely the Application Design and Development Technology, Service Access domain, Service Integration domain and the Application Portfolio.

The Application Design and Development Technology domain describes the recommended application design and development methodology and technical standards. Standardisation on the use of application technologies is necessary so that common business functions and information can be shared amongst government agencies. Application technology standardisation is also a fundamental requirement before government agencies can integrate their various functions as a seamless government service to the citizens and commercial establishments.

Service Access domain defines the technology categories, technology components and associated standards for access channels that allow users to interact with the requested applications and for the applications to communicate responses to the users. It highlights key architecture design considerations and recommends best practices for service access implementation.

Service Integration domain describes defines the various service integration technology categories, technology components and associated standards. It highlights the key architecture design considerations and recommends best practices for implementing service integration solutions.

The Application Portfolio describes the Information and Communication Technology (ICT) solutions that can improve integration of government services through sharing and re-use of applications and their components amongst government agencies. The application portfolio is used to explore and find opportunities for application sharing, re-use and improvements.

1.2 Objectives and Benefits of SRM

The SRM is the main component of Oman eGovernment Architecture Framework (OeGAF) that describes the ICT solutions that provides technology standardisation and service integration so as to improve and enhance government functions. The SRM aims to encourage the re-use of applications and their components to derive economies of scale for the Oman Government.

The SRM analyses current limitations and gaps, and lists the opportunities for consolidation and integration of ICT solutions. Besides cost efficiency, the recommendations from the SRM would enable government agencies to provide more integrated government services to the citizens, residents and commercial establishments. With the SRM, the Oman government agencies would be more responsive to the needs of the citizens, residents and commercial establishments in providing convenient, integrated and faster turnaround services.

With proper follow through, the SRM will bring about the following potential benefits:

(a) Improve government services to stakeholders (citizens, residents and commercial establishments)
(b) Enhance interoperability across government agencies
(c) Leverage on current ICT investment and assets; and reduce Oman Government's total cost of ownership on future ICT investments
(d) Align agencies ICT projects to shared services and central ICT initiatives.

1.3 Background on Oman eGovernment Architecture Framework

OeGAF consists of four main architectures as follows:

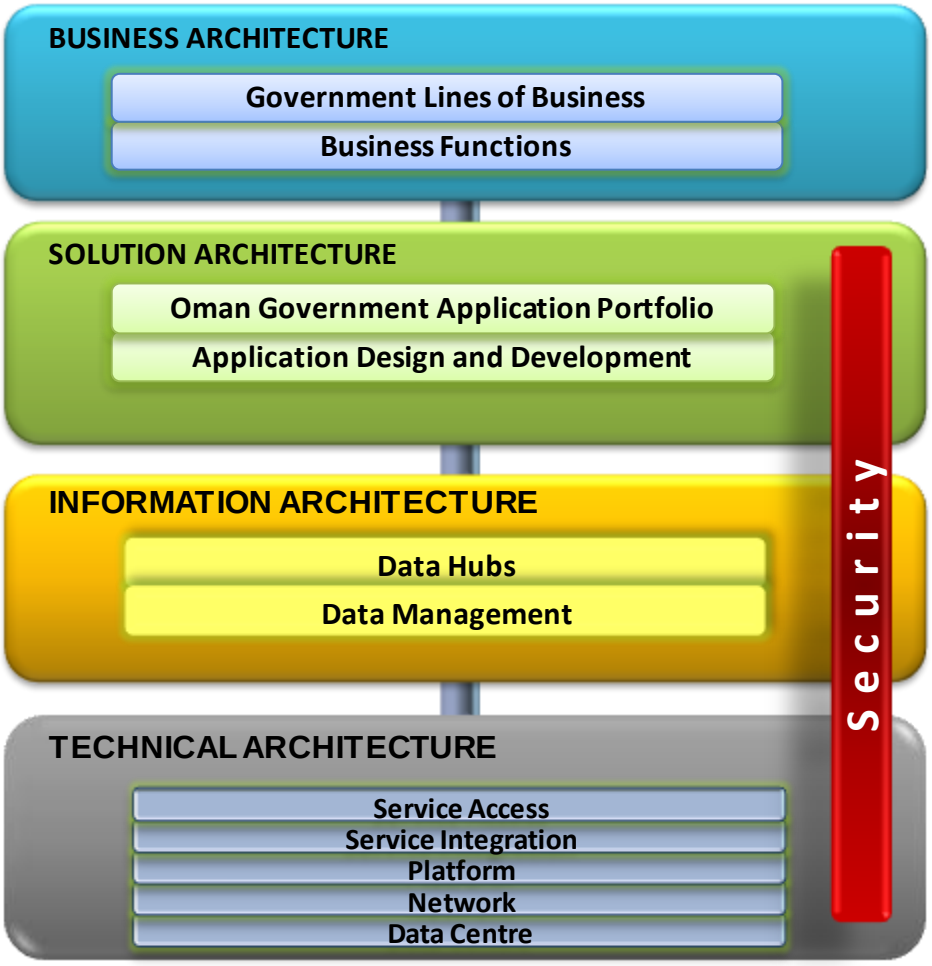
(a) Business Architecture

(b) Solution Architecture

(c) Information Architecture

(d) Technical Architecture

Each of the architecture has a corresponding Reference Model. Each Reference Model describes a framework to define and organise the architecture elements. Figure SA-1 depicts the overall OeGAF Reference Architecture.



(need to replace diagram)

Figure SA-1: OeGAF Reference Architecture

The OeGAF four Reference Models are:

The **Business Reference Model (BRM)** describes the different lines of business and the associated government functions of the Oman Government that cut across the boundaries of different agencies.

The **Solution Reference Model (SRM)** describes the common applications and application components that can be shared across the Oman Government. It includes the technical standards and security considerations pertaining to the design and implementation of solutions and applications.

The **Information Reference Model (IRM)** lists the data definitions and data elements of common and shared data that are used across the Oman Government. As part of the initial baseline scope, IRM describes the data pertaining to 'Person', 'Establishment' and 'Land' data hubs which are commonly used by various agencies' applications. It also defines technical standards, design and security considerations and best practices related to the management of data.

The **Technical Reference Model (TRM)** defines the infrastructure technologies and their respective technical standards to enable better system integration and interoperability across the Oman Government. It also defines the security considerations and standards related to the infrastructure technologies.

The mention of the technologies and technical standards in the TRM, IRM and SRM is to provide logical and easy reference for readers.

1.4 Relation to Other OeGAF Reference Models

The SRM is the main component of OeGAF that links government functions, ICT solutions, data hubs, technologies and technical standards. The relations of SRM to the other reference models are as follows:

(a) The SRM follows up on the recommendations by the BRM for consolidation and integration of ICT systems to support the streamlining, integration and optimisation of business processes. For example, through the BRM, there is a need for a One-Stop-Shop system to be enhanced that allows online application of all commercial establishment licenses. The SRM provide a more detailed ICT solution description on the One-Stop-Shop system.

(b) The lines of business and government functions defined in BRM will aid the discovery of data required by the government functions that will result in the creation of conceptual data model where the following central repositories can be easily identified with detailed definition in IRM:

(i) Central repository of data on citizens and residents required to support the Civil Event Records Maintenance function (Person Hub)

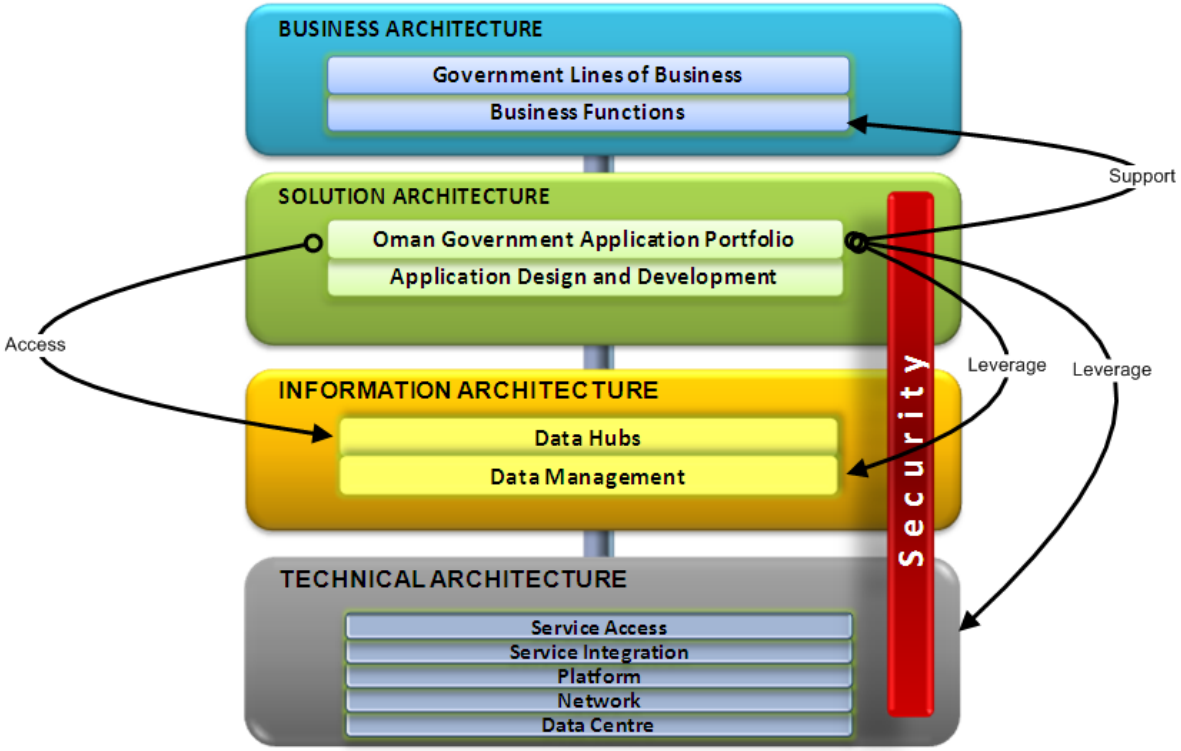
(ii) Central repository of data on commercial establishments required to support the Licensing and Regulatory Control of Commercial Establishments function (Establishment Hub)

(iii) Central repository of data on land information required to support the City Planning and Development function and Land, Building and Public Facilities Development and Management function (Land Hub).

The SRM references the IRM's data hubs and data management technologies for the development and deployment of applications to support the above government functions.

(c) The SRM leverages on the TRM's infrastructure technologies, technical standards and central infrastructure initiatives such as the Oman Government Network and the Oman eGovernment Services Portal.

Figure SA-2 below provides a pictorial overview of how the SRM is related to BRM, IRM and TRM.



(need to replace diagram)

Figure SA-2: Relation to Other OeGAF Reference Models

1.5 Scope of Solution Reference Model

As described above, the SRM has three main architecture elements – Application Design and Development Technology domain, Service Access domain and Service Integration domain.

The Application Design and Development Technology domain provides the following information:

(a) A standard application development methodology that describes the various aspects of application development and deployment life cycle
(b) A list of architecture design factors for consideration to aid agencies in application development
(c) A list of mandatory and recommended technical standards for agencies to comply, and
(d) Best practices in the various steps of application development.

The Service Access domain provides the following information:

(a) A list of possible service channels that the public and government employees can access
(b) A list of architecture design factors for consideration to aid agencies in selecting the right access channels
(c) A list of mandatory and recommended technical standards for agencies to comply, and
(d) Best practices in service access provisioning.

The Service Integration domain provides the following information:

(a) A standard application development methodology that describes the various aspects of application development and deployment life cycle
(b) A list of architecture design factors for consideration to aid agencies in technical integration
(c) A list of mandatory and recommended technical standards for agencies to comply, and
(d) Best practices in the various integration methods.

The first version of OeGAF provided the Application Portfolio for the Oman Government. In essence, the Application Portfolio is an updated list of the main applications in the government agencies. The Application Portfolio reviews how current applications support the government functions and how common data are accessed by the applications. The Application Portfolio, upon careful analysis, also recommends the target state comprising of various ICT solutions to improve the integration of government services.

As ITA would continue to update the Application Portfolio, it is no longer considered as an important architecture element in the SRM. ITA, however, would use the Application Portfolio for references before providing recommendations and advices to government agencies.

1.6 Structure of SRM

As shown in Figure SA-3, the SRM consists of three main domains – Application Design and Development Technology, Service Access and Service Integration domains.

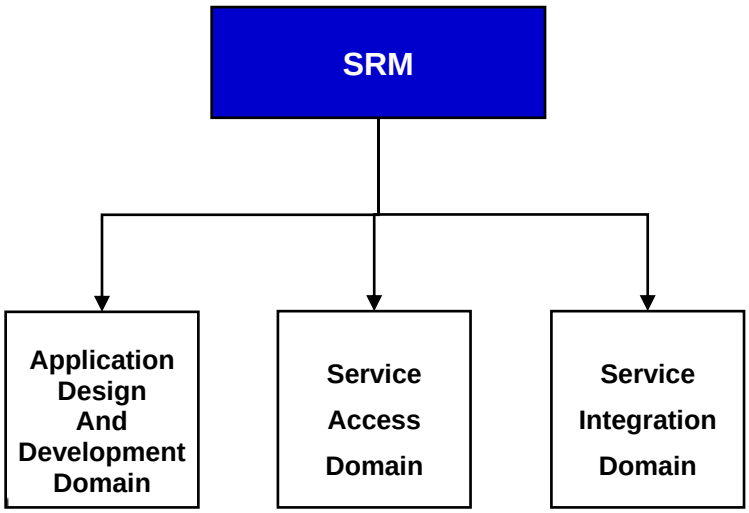


Figure SA-3: Structure of SRM

The Application Design and Development Technology Domain addresses the following:

(a) How to design, develop and test the applications?

(b) What are the application standards to comply with?

(c) What are the application design considerations and best practices?

The Service Access Domain addresses these questions:

(a) How and what channels users can access to eServices and information?

(b) How applications communicate responses to users?

(c) What are the design considerations and best practices for service access implementation?

The Service Integration Domain addresses the following:

(a) What are the various service integration technology components?

(b) What are the various service integration standards?

(c) What are the design considerations and best practices for implementing service integration solutions?

The contents of the three technical domains are structured as follows:

(a) Intent

Describes the intent of the technology domain

(b) Domain Design Principles

Describes the domain design principles

(c) Technology Categories and Components

Lists the technology categories and their components

(d) Architecture Design Considerations

Describes the key design considerations when developing an application

(e) Technical Standards and General Standards

Lists the mandatory and recommended technical standards for application development and deployment

(f) Best Practices

Describes the international best practices in application development

(g) Obsolete Technologies

Describes the list of obsolete solution technologies that government agencies have to comply (i.e. government agencies cannot use these obsoleted technologies).

The Application Design and Technology Domain will have an extra section below:

(a) Application Development Methodology

Describes the recommended application development methodology for adoption by government agencies.

1.7 Target Audience

The SRM is a reference that provides an insight to both ICT solutions and technical-oriented application technology domain. The target audience for the SRM are as follows:

(a) ICT Planners / Architects (ITA and Agencies)

The SRM together with the BRM and IRM can be used by the strategic planners and architects in ITA to identify potential shared / common solutions for the use by all government agencies. On the other hand, government agencies can use the SRM to guide their planning and development of new or revamped ICT systems. In addition, for agencies developing their very own specific solution reference models, the SRM is an important reference that can help agencies to architect its own set of solutions that are aligned with OeGAF.

(b) ITA Project Managers

The project managers in ITA can reference the SRM in planning and managing the central initiatives and projects for the Oman Government.

(c) ICT System Owners in Agencies

ICT system owners can reference the SRM to leverage on the shared services or central initiatives. By knowing the availability of common systems, system owners can avoid building duplicate systems.

(d) ICT Directors / Managers in Agencies

The ICT directors and managers can reference SRM to understand the best access channels for the public, while considering the efficient ways of application integration. The SRM can help in the strategic planning process and identify potential shared / common applications across agencies or within the agency. They must also comply with the application technology standards in the development of their applications.

(e) ICT Vendors

ICT vendors should use this document to propose and implement ICT systems that comply with the standards stated in the three domains. ICT vendors can also refer to SRM to architect government agencies' application architectures.

1.8 SRM Design Principles

Design principles, which describe the preferred directions, aspired attributes and practices, are required to guide the development of the architecture. The following are the overarching Solution Architecture design principles while specific domain design principles for the three domains are respectively described within the domain sections.

Principle 1: Use Current Appropriate ICT Solutions to meet the Government Business Needs and Operational Requirements

ICT solutions need not necessarily be new. Current and appropriate ICT solutions, where possible, should be leveraged to meet business needs and budgets.

Principle 2: Optimise and Share Government ICT Solutions for Cost-Effectiveness

There are many current solutions used amongst the Oman government agencies. A sound working solution in one government agency could potentially be used in the other agencies. The sharing or re-using of these solutions would lower the overall Total Cost of Ownership for the Oman Government.

Principle 3: Design for Serviceability, Reliability, Availability and Scalability

Developing and maintaining government-wide solutions require thorough planning and design. Solutions have to be easily serviceable (i.e. easily change or amend when necessary), reliable (i.e. perform the function as promised to users), available (i.e. the service is online as stipulated to users), and scalable (i.e. can support the increase in number of users or transactions). These features will provide performance and confidence amongst users accessing government services 24x7 (access anywhere and anytime).

Principle 4: Promote Agility and Quality in the Government ICT Solutions

The government solutions have to be agile to support the fast and constant changes to government business requirements. Quality solutions ensure that the service level business requirements are met. These government solutions are effective when they can be quickly adapted, without compromising on quality, to meet the changing demands.

Principle 5: Ensure Security in the Development, Implementation and Management of Government ICT Solutions

The government solutions have to be secured. With increased service delivery over the Internet, security considerations and standards have to be in place. Any security lapses will result in bad reputation, mistrust and incur damages.

Principle 6: Use ICT Solutions that support Open, Vendor-Neutral Standards and Best Practices with Wide Industry Acceptance

To protect investments in government applications and solutions, it is logical not to be locked-in to one vendor. ICT solutions have to be vendor and platform independent as much as possible. In addition, solutions with wide industry

acceptance should be selected as the vendors have to cater for global requirements and support.

1.9 Governance of SRM

The ongoing management and execution of SRM is part of the overall governance of OeGAF. It includes reviewing and updating the three technical domains.

2 Right Solutioning for Government

2.1 Intent

The intent of this section is to aid government agencies in choosing and implementing the right ICT solutions that support the business functions and processes as defined or prioritised in BRM.

2.2 Role of ICT Solutioning in OeGAF Architecture Vision

Figure SA-4 below illustrates the OeGAF Architecture Vision. In BRM Section 3, it was stated how the government functions and processes will drive the implementation of relevant ICT solutions, information/data and infrastructure to deliver quality eServices to the beneficiaries.

Similarly in BRM Section 3, there is a set of guidelines on “Improving Performance of Government Functions”. After the identification of the prioritised business functions and improved business processes, SRM aims to provide options for the “*Presentation*” and “*ICT Solutions*” layers as highlighted in the figure below. This section, therefore, describes the importance of choosing the appropriate ICT solutions. Government agencies need not use the latest technologies, but rather to implement the right ICT technologies and methodologies that meet the business requirements.

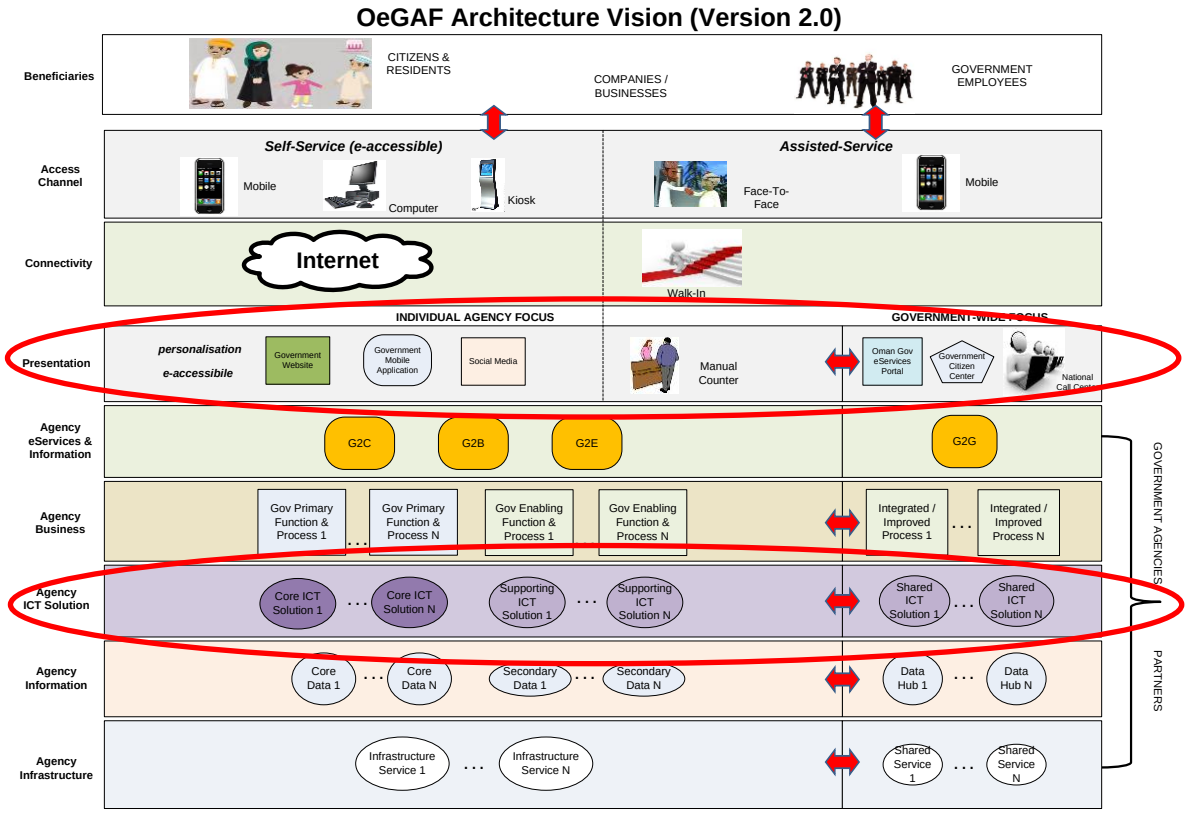


Figure SA-4: OeGAF Version 2.0 Architecture Vision

2.3 Guide for Government ICT Solutioning

The following steps will guide government agencies to implement the right ICT solutions:

(i) Understand Your Business Requirements

It is important to start by understanding the actual business requirements. Government agencies have to prioritise their government functions and sub-functions, and to carry out a set of selective business process improvements. The outcomes of these prioritised government functions and business process improvements would lead to the actual business requirements.

(ii) Develop Solutions at Presentation Layer

In order to reach the public and government employees through the various access channels, each government agency has to provide the following solutions:

(a) Counter Service

Currently, most services are provided via the counter (or face-to-face). While counter service will continue, government agencies must attempt to reduce them as much as possible. As part of the Oman eGovernment Transformation Plan, government agencies have to migrate and automate their counter services into eServices where possible.

(b) Government Agency Website

Information and eServices, as much as possible, can be delivered over the government agency's website. As according to ITA's policy, government agency website has to support both Arabic and English.

(c) Government Mobile Application

Each government agency should consider providing its information and eServices over mobile applications. The government agency mobile application must be easily downloaded with regular updates.

(d) Social Media

To be effectively connected with the public, government agencies should consider implementing formal pages in the social media. Latest information can be easily pushed to the public, while comments and discussions can also be carried out.

(e) Leverage on Government-Wide Shared Services or Central Initiatives

From a government-wide perspective, government agencies should consider leveraging on the currently available shared services or central initiatives. This would speed up implementation and aid government-wide integration. Please refer to Shared Services and Central Initiatives for more details.

(iii) Develop General eServices (General eServices Layer)

When we zoom into the 'Agency ICT Solution' layer of the OeGAF Architecture Vision, we get the details as shown in Figure SA-5. Instead of developing solutions for each application or each eService, it is highly recommended that government agencies develop their very own general re-usable eServices. As part of Service-Oriented Architecture (SOA), these general eServices are built once but can be re-used for numerous times. Government agencies are recommended to leverage on potential government-wide general eServices.

Examples of Individual Government Agency general eServices are:

(a) Search Service

It is a common need for any user - either through a counter service, a webpage, a mobile application or a call centre – to search for specific information. Instead of developing search for each application, it is recommended that government agency develops its own search service.

(b) News & Events Service

This eService is to provide the relevant updates relating to news or events. When there is specific news or events, the information can be broadcasted to the users via their preferred communication channel such as email, SMS, website link, social media link or application.

(c) ePoll / eSurvey Service

A useful eService to carry out various polls and surveys.

(d) eHelp & FAQ Service

This would be a common eService to help the user and to address frequently-asked questions (FAQs).

(e) Social Media / Sharing Service

An important service that provides common links to the various government agency's social media pages.

(f) Feedback Service

Another important service to receive feedback, comments and complaints.

OeGAF Architecture Vision (Version 2.0)
Oman Government Solutions

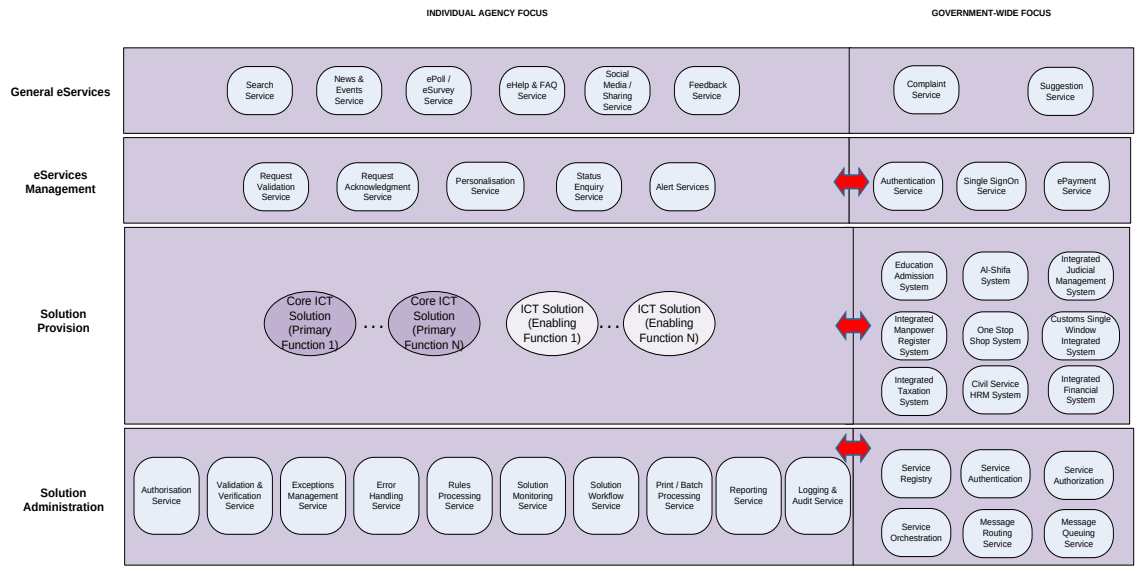


Figure SA-5: Oman Government Solutions

(iv) Develop eServices Management Solutions (eServices Management Layer)

Similarly like the above, a suite of management eServices can be developed. As part of SOA, these management eServices can be called or referenced by the actual eServices. Government agencies should leverage on government-wide eServices Management modules where available.

Below is the recommended set of management eServices to be developed by each government agency:

(a) Request Validation Service

During the actual eService transaction, there will be a need for validation of data entered by user. By calling this service, certain validations can be carried out such as checking the validity of Citizen ID, Government Employee ID/Number, CR Number and even Credit Card number format. In addition to these validation examples, each government agency should develop other in-house validation services for data with the highest number of transactions.

(b) Request Acknowledgment Service

Upon the completion of data validation, this service is to acknowledge that the user has submitted an eService request. This service could also inform the expected eService process duration and other related information. Below is an example of an eService acknowledgment.

> Your service request has been validated and received. We will process your request and respond back to you within *N* days/hours. You could also check the update status of your request using the Request Number {XXX-DDMMYYYY-NNN}. Thank you.

(c) Personalisation Service

Over time as eService is common, users will demand for some personalisation service. This is a value-added service that will delight or please the users. Examples of the personalisation

parameters include preferred communication channel (email / SMS), preferred website landing page, preferred city / town for interaction and preferred ePayment mode.

(d) Status Enquiry Service

For certain transactions, there is a need for users to track the status of their eService requests. Hence, this service would allow the users to do a self-service enquiry using the Request Number.

(e) Alert Service

On certain occasions, users want to be alerted or notified when a specific condition developed. For example, when the eService request has been processed, a payment will automatically be carried out. With the alert service, the user will be informed that the request was completed and a payment will be deducted from the user's preferred bank account.

(v) Develop General Re-Usable Administrative Application Services (Solution Administration Layer)

As part of good SOA practice, it is more efficient and effective to implement a suite of re-usable administrative application services. In essence, these services will be highly re-used by the various applications resulting in faster application development and implementation. Government agencies are recommended to leverage on government-wide administrative re-usable application services where available.

Below is a list of the recommended re-usable administrative application services to be developed by government agencies:

(a) Authorisation Service

A central service that authorises internal and external access to applications owned by the government agency. Note that this service is normally carried out after the authentication service.

Government agencies are encouraged to leverage on the government-wide authentication service for the public and government employees.

(b) Validation & Verification Service

This service carries out the typical validation of data entered into the application system – for example date entered must be in DD / MM / YYYY format. It also verifies the data entered – for example the tourist visa request date entered has to be greater than the current date but not greater than a year from the current date. It is recommended to develop many highly re-usable validation and verification checks. Note that this service is called via an application where it would eventually be used as part of the validation in the eService.

(c) Exceptions Management Service

While many rules and conditions can be computed and programmed, there will always be exceptions. Thus this is a useful service to manage exceptions.

(d) Error Handling Service

It is important that users get informative and consistent error messages. Depending on the error, this service would provide the actual respective error message(s).

(e) Rules Processing Service

It is not a good programming practise to ‘hard code’ the processing rules into the programs or solutions. Instead, with the use of commonly defined rules and conditions in a central repository, this service can read the repository and process the request accordingly. This service allows business requirements to be dynamic.

(f) Solution Monitoring Service

There is always a need to monitor the health status of a solution or application – in particular critical application during peak periods. This service monitors the solution status and updates to the various interested parties such as ICT staff, ICT vendors and business owners.

(g) Solution Workflow Service

As ICT solutions have to support the business processes, it is common that workflows have to be automated and monitored. This service allows workflow to be defined and integrated.

(h) Print / Batch Processing Service

This service provides print and/or batch processing. For example, at the end of each month, all the government employees' salaries are processed, printed into e-salary and distributed securely to each individual government employee.

(i) Reporting Service

When applications are built in silos, they would develop their own reports. This service, on the other hand, would aid the integration of data and reporting that would be useful for business owners and management staff.

(j) Logging & Audit Service

All eService and application transactions have to be logged. This service provides standard transaction logging and allows access to the transaction logs for audit purposes.

(vi) Implement Core ICT Solutions (Solution Provision)

Government agencies are highly encouraged to firstly develop and implement core ICT solutions that would automate or improve their primary functions and processes. This would be followed by the

implementation of ICT solutions that support their enabling government functions and processes. Government agencies are also recommended to leverage on government-wide ICT solutions where possible. Note that ICT automations are implemented only after the business processes have been reviewed and improved as recommended in BRM.

3. Application Design and Development Technology Domain

3.1 Intent

The Application Design and Development Technology Domain defines the domain design principles, application development methodology, application technology categories, technology components and associated standards. It highlights key architecture design considerations and recommends best practices.

3.2 Domain Design Principles

The following are the Application Design and Development Technology Domain design principles:

Principle 1: Reduced Application Complexity

While applications are required to automate the business functions and requirements, the applications developed need not be complex. Reducing the application complexity, or making the application as simple as possible, is regarded as a well-designed application.

Principle 2: Implementing Widely Adopted Standards can Facilitate the Adaptability, Reuse and Cost-effectiveness of Applications

Applications have to be adaptable to meet the changing business requirements. Reuse components of applications will allow faster application development. The use of widely adopted standards in application design and development assures that the application is modular and well-structured which will result in the adaptability of the application to easily change, allow application reuse and results in cost-effectiveness as the effort for application development and maintenance reduced.

Principle 3: Ensure Security in the Application Design and Development

Security has to be considered at the onset from application design. Security is not an end-product, but rather it should be integrated in the application development lifecycle.

3.3 Application Development Methodology

3.3.1 Types of Application Development Methodologies

There are broadly two types of application development – the waterfall and iterative methodologies.

The waterfall methodology focuses on structured progression at the defined phases of application development. Each phase consists of a definite set of activities and deliverables that must be accomplished before progressing to the next phase. With the waterfall approach potential changes can be easily analysed, large distributed teams can be coordinated, and project budgets are predictable as there are defined phases. On the other hand, the weaknesses of the waterfall approach include lack of flexibility, difficulty in predicting actual outcomes for the software, discouragement of team cohesion, and the tendency to not discover design flaws until the testing phase.

Iterative methodology focuses on building a highly skilled and tightly knit team that stays with the project from beginning to end. The formal project deliverables are commonly the actual working software and the essential system documentation that is completed at the end of the project. Rapid feedback from users increases the usability and quality of the application, early discovery of design flaws, an ability to easily roll-out functionality in stages, a more motivated and productive team, and finally knowledge retention for the duration of the project are often observed from this approach. However, drawbacks such as difficulty in coordinating large projects, the possibility for the project to never end, a tendency to not thoroughly document the system after completion, and the difficulty in predicting exactly what features will be possible within a fixed time or fixed budget. Iterative methodology may take various

forms such as agile programming, extreme programming (XP), rapid application prototyping and Scrum.

Both the waterfall and iterative methodologies can be used for application design and development. Appropriate mixture of both approaches can often produces benefits and success in project. The right choice of application methodology would depend on a case-to-case basis; however it is common to have a mixture of the two methodologies for large or complex application development projects.

3.3.2 Programming Paradigm

Procedural programming involves writing code that is as concise as possible and coding directly for the end result. Most procedural programming uses targeted groups of functions that immediately address the problem at hand. If the project grows large, the developer or developers could end up having to maintain a large number of individual functions, and in some cases, the logic of different functions can become confusingly similar and difficult to maintain.

Object-Oriented programming (OOP) focuses on abstract relationships and an hierarchy of related functionalities. Similar functionalities can share common codes, making maintenance much easier. Code reuse is increased as well, as ease to adapt the abstracted base functionality for new tasks. OOP also can aid in large-scale program design, helping encapsulate and categorise the different sets of functionality required by each part of the system.

OOP techniques can help software development in the following ways:

(a) Abstraction
    (i) Modelling of real-world objects makes it easier to describe and communicate behaviour
    (ii) Objects are a natural way of creating, packaging and using software components

(b) Encapsulation of knowledge means that behaviour can be isolated; this in turn means that changes in requirements can be accommodated without affecting the entire system

(c) Inheritance allows us to re-use objects that have already been created

(d) Polymorphism provides the ability to hide many different implementations behind a single interface

While both procedural and OOP can be implemented, modern programming languages, in particular for mobile application development, focus on OOP. Refer to Appendix C for the details of OOP.

3.4 Technology Categories and Technology Components

Application Lifecycle Management integrates design, development, testing, deployment and configuration management as shown in Figure SA-6. It cuts maintenance time by synchronising application design and maximises investment in skills, processes and technologies.

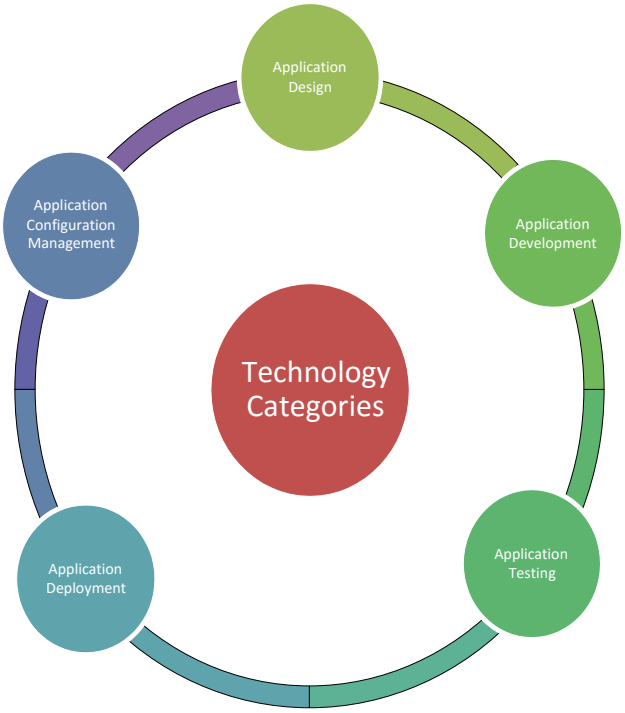


Figure SA-6: Technology Categories under Application Design and Development Technology Domain

As each process in the Application Lifecycle Management uses various technologies, each of these five processes would naturally become a technology category.

Figure SA-7 shows the association between the technology categories, technology components and its relevant standards.

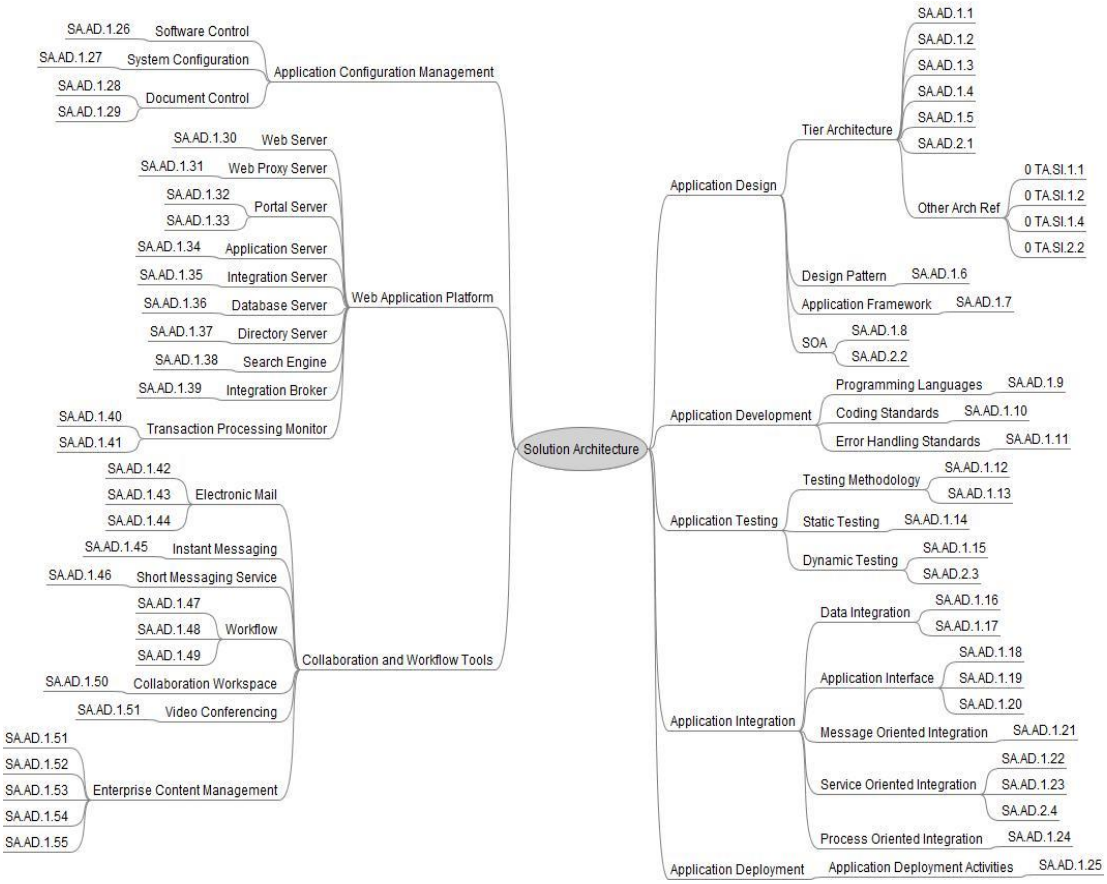


Figure SA-7: Mapping of Categories, Components and Standards for Application Design and Development Technology Domain

Table SA-1 below describes both the technology categories and technology components of the Application Design and Development Technology Domain.

Technology Categories	Technology Components
Application Design	**Tier Architecture** In tier architecture, the software components of an application are structured into "logical" layers or parts. Each tier interacts with only the tier directly below, and has specific function that it is responsible for. Typically, the application is structured into three tiers: (a) Presentation (b) Business logic (c) Data access
	Design Pattern A design pattern is a solution to a frequently occurring software scenario. Software design patterns serve as examples of reusable designs that have proven themselves in practice. A design pattern captures the essential insights, so that others may learn from it, and make use of it in similar situations. Studying patterns of a development situation and its solutions provides a level of problem recognition and a general formulation of a solution. The pattern should be adapted to the context in which it is being used.
	Application Framework A framework is a set of classes and interfaces that co-operate to solve a specific type of software problem. It is these interconnections that provide the architecture and design so that developers can free themselves from building the infrastructure and concentrate on writing codes that extend the

	framework behaviour to suit the business requirements.
	Service-Oriented Architecture (SOA) SOA provides a set of principles or governing concepts used during phases of systems development and integration. Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. SOA separates functions into distinct units, or services, which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications.
Application Development	**Programming Languages** A programming language is an artificial language designed to express computations that can be performed by a computer. There are various generations of languages (2G – 4G), which were developed to meet different requirements and objectives.
	Coding Standards Code written in a consistent manner will be more maintainable and robust. Application development teams should set and follow the coding standards. It also helps new developers coming into the project to easily pick up and understand the codes.
	Error Handling Standards

	Reliability is a major characteristic of high-quality software. During the development, consistent error handling will enhance the quality of the code and produce a robust and resilient application.
Application Testing	**Testing Methodology** Testing methodology refers to a set of rules and practices used to test a system. The cost of defect identification and correction increases exponentially as the project progresses. As such, defects should be identified and corrected early. To do so, testing activities should be conducted throughout the different phases of the software development life cycle.
	Static Testing The process of testing a system or component based on its form, structure, content or documentation is known as static testing.
	Dynamic Testing The process of testing a system or component based on its behaviour during the execution of a computer program is known as dynamic testing.
Application Deployment	**Application Deployment Activities** Activities that make a software system available for use. After the application has been developed and tested, it has to be deployed for production or “go live”. The main activities include configuring the production environment, executing job scripts to

	transfer the application from testing to production environment, and loading initial data into production database.
Application Configuration Management	**Software Control** Identifies the functional and physical attributes of software at various software development phases and performs systematic control of changes to the identified artifacts for the purpose of maintaining software integrity and traceability throughout the software development life cycle. Configurable items include: (a) Application source code (b) Scripts (c) Commercial-off-the-shelf (COTS) software
	System Configuration Documents the infrastructure requirements for application to execute successfully (e.g. enterprise application may require certain ports/services to be opened at agency's firewall, or permanent TCP connection is required to be established across network segments) Configurable items include: (a) Operating System (OS) (b) System patch
	Document Control Identifies and controls project documentation (e.g. project plan, schedule and budget) that will change

	during the software development life cycle. Configurable items include: (a) Project plan, schedule and budget (b) Requirement specifications (c) Design documents (d) Functional specifications (e) Test plans and results (f) User and administrator guides (g) Project process and procedures (h) Project standards

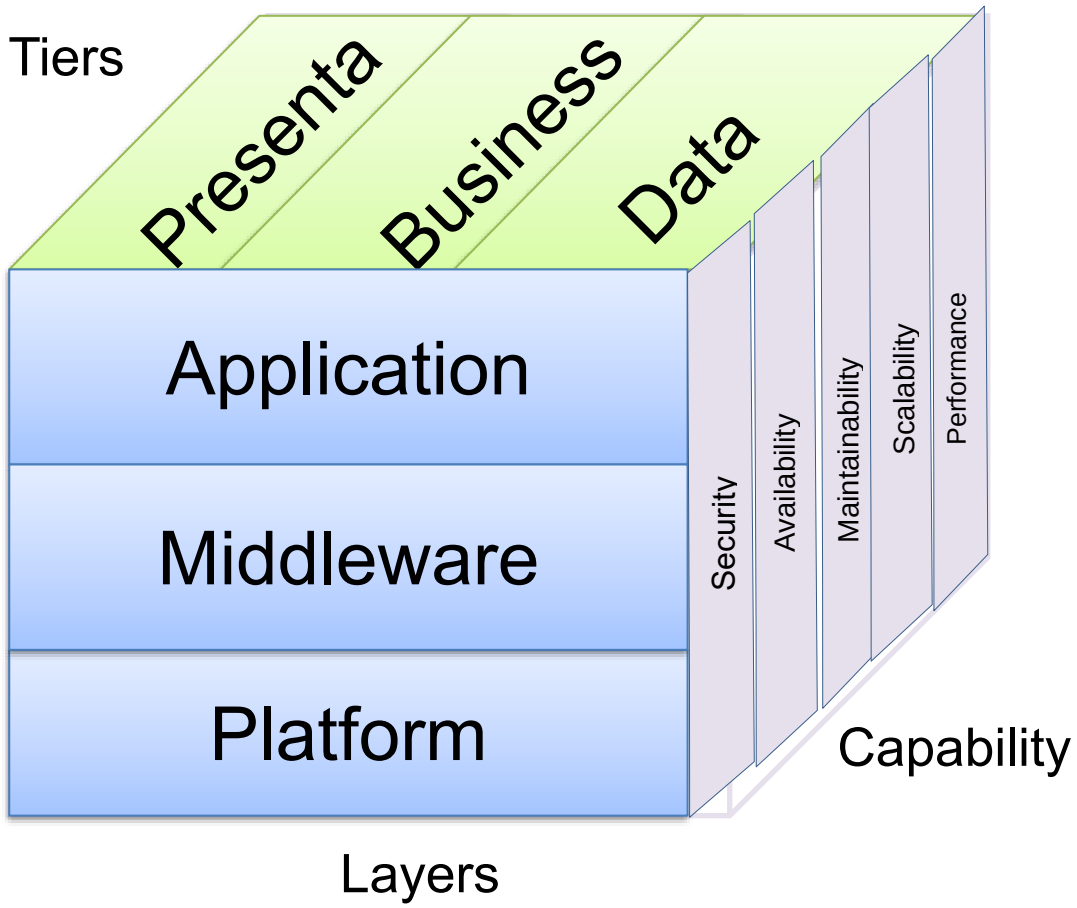
Table SA-1: Application Design and Development

Technology Categories and Components

3.5 Architecture Design Considerations

3.5.1 Application Design

Application design and development tools and technologies have been evolving rapidly in an increasingly distributed and real-time business environment. Figure SA-8 describes a conceptual framework that illustrates the relationship amongst the various elements of an application. The capabilities, tiers and layers are inherently orthogonal, which make the architecture a collection of tradeoffs and decisions. Hence application design requires all these factors to be taken into consideration.



(need to replace with new diagram)

Figure SA-8: Application Development Conceptual Framework

Layers are abstractions of the underlying implementations of the application. The platform layer is the operating system. The middleware layer is the operating environment such as a web, application or database server. The application layer is the highest level that contains the presentation, business and data logic.

Tiers are logical and NOT physical divisions of a system and are differentiated by their assigned roles and responsibilities. For example, all tiers can exist in one machine or each tier resides in separate machine. A typical application consists of three tiers: presentation, business logic and data access tier. The presentation tier focuses on interacting with users. The business logic tier determines the business process and rules of the application. The data access

tier manages the storage and retrieval of data from data resources such as databases and files.

Capabilities are non-functional, observable system qualities that do not represent specific functions and cannot be satisfied by any one component. These are properties that are observed in a collection of components working together.

3.5.1.1 Tier Architecture

As mentioned earlier, an application can be logically structured into three tiers: presentation, business logic and data access. The way these three tiers are partitioned in an application will give rise to different types of the application design.

Tier Type	Definition
One-tier or Monolithic Application	These are applications where the code that implements the presentation, business rules, and data access are tightly coupled together as part of a single application that runs on a single computer in a non-distributed manner.
2-tier Client/ Server Application (fat client)	These are applications where the presentation and business logic are grouped on the client machine and a shared data source is accessed over a network connection.
2-tier Client/ Server Application (fat server)	These are applications where the presentation is on the client machine and much of the code that implements the business rules are tightly integrated with the data access code, sometimes in the form of database stored procedures and triggers.

Tier Type	Definition
3-tier Client Server Applications	These applications are partitioned into three executable tiers of code: the presentation/user interface which runs on the traditional desktop client, the business rules which reside in the business logic tier, and the business data (including database object) which resides in a data access tier.
Web-enabled Application	These applications are usually partitioned into at least 3 tiers: the user interface which is the browser client, the business rules which reside in the business tier, and the business data which resides in the data access tier.
N-tier Application	An N-tier application refers to an “at least” 3-tier applications, be it a 3-tier client/server or web application. These applications have distinct separation for presentation (including support for different client machines like browsers and mobile browsers), business rules and data access. N-tier applications typically have distributed deployment with multiple physical servers handling load balancing and high availability.

Table SA-2: Tier Architecture

The advantages and limitations of each type of application tier are listed below in Table SA-3. The considerations are based on the capabilities of the

application types and it includes security, availability, scalability, maintainability and performance.

Tier	Advantages	Disadvantages
One-tier or Monolithic Application	• Very high performance • Self-contained • Easier to secure • Easier to manage	• Potentially costly and time consuming to modify due to the nature of the programming language used. • Difficult to share services and data • Little reuse of code between monolithic applications. • Can be accessed using only a single user interface
2-tier Client/ Server Application (fat client)	• Good performance as application logic runs within clients • Fewer network traffic	• Difficult to deploy, as every client needs to be updated when changes occur • Business rules on the client potentially expose business rules to users, resulting in compromised confidentiality • Difficult to reuse application logic as it tends to bound tightly to the presentation logic

Tier	Advantages	Disadvantages
2-tier Client/ Server Application (fat server)	• Improved security is possible by restricting access only to the stored procedure and denying access to the data • Reuse is improved by enabling multiple presentation components to call the same stored procedures	• Stored procedures or triggers have limited capabilities in handling business logic. • Stored procedures are inherently more difficult to maintain due to its procedural nature. • Long running stored procedures tie up database connections. • Stored procedures are tied to a particular vendor, and cannot be moved easily to a different database
3-tier Client/ Server Application	• Allows for greater reuse of component. • Easier to modify when business rules change, resulting in improved maintainability. • More flexible in terms of scalability as the workload increases	• Reusable components are more complex to design and develop. • Performance may be slower due to the data transfer and method invocations across multiple tiers. • Skills set requirements are more complex
N-tier Application (inclusive of Web Application)	• Similar to 3-Tier Client/ Server Application	• Similar to 3-tier client-server applications

Tier	Advantages	Disadvantages
	• Allow greater flexibility to change and scale at each layer • Support different types of user interfaces e.g. Web Access Protocol (WAP) for mobile browser	

Table SA-3: Comparisons of Application Tiers

Areas of consideration when selecting tier architecture for application design:

(a) Number of Users

If the number of users accessing the application is large or cannot be estimated, consider using 3-tier or N-tier application. This is because it is easier to scale up the application with 3-tier or N-tier architecture.

(b) Network connectivity of the users

One-tier or two-tier usually require certain ports and settings on the network. This is only feasible if the users are in the controlled environment (for example, Intranet). If the users are on the Internet, consider going for 3-tier web application instead.

(c) Cost and time for the new application

Generally, a 3-tier application will cost more and take more time to develop.

(d) Ease of deployment

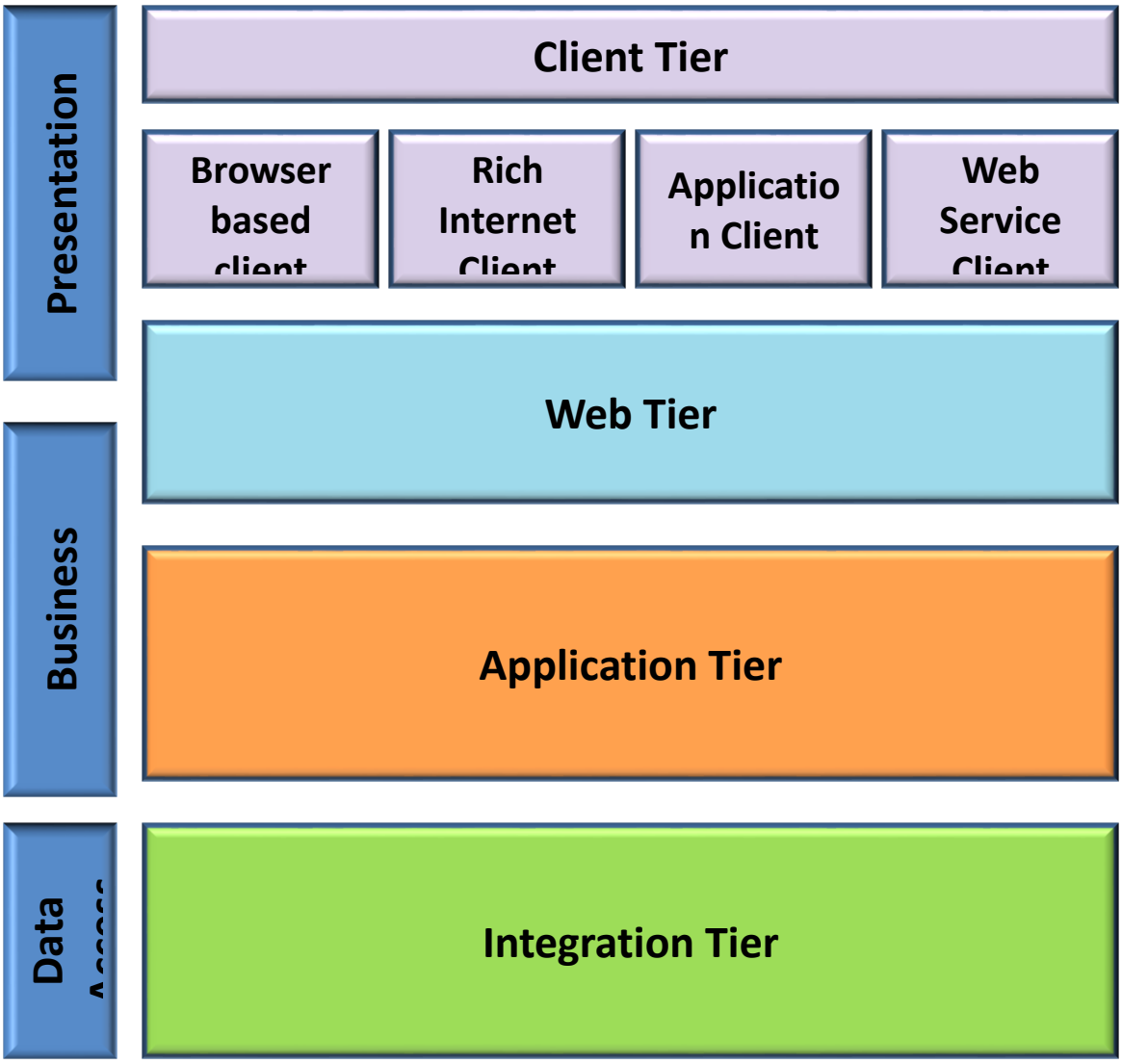
Applications utilising thin client (e.g. browser-based) are easier to deploy as compared to application utilising 2-tier client/server.

(e) Requirements for application to share services and data

It is usually easier to share services and data using 3-tier or N-tier architecture.

Components of Distributed N-tier Applications

Figure SA-9 shows the components that should be taken into account during application design for a distributed N-tier application.



(need to replace with new diagram)

Figure SA-9: Technology Components of an N-tier application

(a) Client Tier

The client provides a means for user to interact with the application. The different types of clients are listed in the Table SA-4 below. These clients differ in the way of access, deployment process and technology used.

Client Type	Description
Browser-based Client	Static or dynamic pages displayed on the web browser. Most widespread use in the industry. Commonly known as thin client.
Rich Internet Application (RIA)	RIA application offers a richer and more responsive web interface (such as drag-and-drop, animation, and video) as compared to the browser-based client. The technologies include binary downloads/plug-ins such as Applets and ActiveX, or lightweight scripting framework such as AJAX and Adobe Flex.
Application Client	Graphical User Interface running in the client machine. Usually created with Visual C/C++, Visual Basic, and Java AWT/Swing. Able to access business tier components directly. Good for highly interactive usage. However, need to update changes for every client station.

Table SA-4: Client Types

(b) Web Tier

The web tier enables the client tier to communicate and interact with the application and integration tiers. It is responsible for managing content selection based on application logic and session state. The web tier could communicate with an application server or directly with the integration tier.

Web tier codes such as ASP, JSP and servlets can be used for low-level application functions such as:

(i) Simple business logic

(ii) Controller component that manages selection of view

(iii) Dynamically generating binary data such as images or new content type.

(c) Application Tier

This tier is mainly responsible for implementing the business rules for the application. It is designed to provide a more scalable, fault tolerant and highly available architecture. However, in some cases, the middle tier may not implement the full business logic. It either works with or delegates the business logic processing to one or more existing systems.

(d) Integration Tier

This tier is responsible for providing data access to the enterprise resources such as file systems, databases, directory service, mainframe, Enterprise Resource Planning (ERP), or any other legacy systems.

3.5.1.2 Design Pattern

Design pattern is a standard solution to a common problem. Instead of designing and building a totally new solution to meet the application requirements, design patterns can be adopted and customised to provide a fast, reliable solution. Design patterns can be categorised as façade (i.e. presentation), account access (i.e. account maintenance and logging in/out), self-service (user interacting with the application typically as an e-service), integration (such as application and data integration) and information aggregation (i.e. aggregating information from various sources to be used for analysis).

Areas of consideration in selecting design patterns are:

(i) Choose design patterns to solve the key business issues.

(ii) Design patterns should be product independent.

3.5.1.3 Application Framework

A framework is a set of classes and interfaces that co-operate to solve a specific type of software problem. It is these interconnections that provide the architecture and design so that developers can free themselves from building the infrastructure and concentrate on writing codes that extend the framework behaviour to suit the business requirements. Examples of widely adopted product-independent application frameworks are Struts and Spring.

A framework has the following characteristics:

(a) A framework is made up of multiple classes or components, each of which may provide an abstraction of some particular concept. These components and services are application-neutral, meaning they support the needs of an application without making any assumptions about it

(b) The framework defines how components should work together based on design patterns and best practices

(c) The framework provides an internal architecture that when leveraged, will provide application reusability across the enterprise, not simply component reusability.

Areas of consideration when evaluating an application framework:

(a) Soundness of the framework. Some of the factors to consider are:
    (i) Extensibility of the framework. Generally the framework should be flexible to accommodate additional requirements arising from the application
    (ii) Ease of maintenance

(b) Quality of code (if available) to access the cleanliness, efficiency and correctness of the implementation

(c) Support by component developers to build in reuse components

(d) Quality of the documentation

The documentation should be comprehensive and sufficient for one to understand and utilise the framework.

(e) Product or project status e.g. vendor financial status, acceptance and support in the developer community.

3.5.1.4 Service-Oriented Architecture (SOA)

SOA is a software architecture that builds a topology of interfaces, interface implementations and interface calls. It is a relationship of service producers and service consumers. SOA offers an evolutionary approach in enabling distributing computing across the internet-connected enterprise. It has brought about substantial impact in the areas of application design and development, application integration and business process management. Government agencies, in particular those with many applications, are to consider adopting SOA.

Areas of consideration when designing an SOA service includes:

(a) Use meaningful service names

This will help the developers to identify the services and operations that they need to use to implement the business processes. The names should be meaningful in the domain of expertise of the service consumer, favouring business concepts over technical concepts. “ValidateID” is an example of a service that validates a given ID.

(b) Select a well-chosen granularity for services

By granularity, we are referring to the number of operations a service should have. For example, let us assume that ProcessRegistration service has a few operations such as ValidateID, ValidateRegistration, CheckAvailability, RegisterRequest and AcceptPayment. The following factors can be considered when designing services:

(i) Services should generally be the unit of testing and release

If granularity is too coarse (i.e. a large numbers of operations are grouped together in a single service), then the number of

consumers for the service tend to large. Hence if any amendment to some aspect of a service is made, perhaps for the benefit of only a subset of consumers, the whole service must be re-released and hence potentially impact all consumers. In the above example, it can be seen that ProcessRegistration is too coarse; the services should be offered at one level lower i.e. ValidateID, ValidateRegistration, etc.

(ii) Select service granularity with the service consumer in mind

One challenge for the service consumer is to find the correct operations to use. Typically the consumer needs to browse a list of services and, having identified a suitable service, the list of service operations. In the extremes of service granularity - either many services with few methods or few services with many tens or hundreds of operations - will tend to impede consumability. The above are examples of services with reasonable granularity where each service has a few operations.

(c) Services should be cohesive and complete

The interfaces created should be functionally cohesive, i.e. a set of operations that belong together because of their function. One way to assess if an operation should be included is to look at the service from the perspective of the service consumer. The naming convention of the services and operations also help us to focus on the functional cohesiveness of the interface, when we ask the question “Is the verb something that the noun does?” For example, in ValidateRegistration service, the registration is being validated as part of the overall registration process

(d) Services should have stateless interfaces

For the services to be resusable, scalable and ready for deployment in high availability infrastructures, the services should not be stateful, i.e. there should not be any reliance on any long-lived relationship between

consumer and provider nor an operation invocation implicitly relying on previous invocation.

3.5.2 Application Development

Application development is required to produce applications that support the business functions. While there is a growing trend in using Customised-Of-The-Shelf (COTS) applications, the demand in the development of bespoken applications and application reusable components continue to be very high.

3.5.2.1 Programming Languages

The choice of tier architecture (i.e. one-tier, 2-tier, 3-tier or N-tier) affects the selection of programming language(s). The programming language(s) has to support the tier architecture efficiently. In addition, the following are areas of consideration when selecting programming languages:

(a) Widely adopted programming language

The programming language has to be widely adopted globablly to ensure that it is reliable and has sufficient vendor support. A matured programming language would have been improved over time for efficiency and removal of programming bugs

(b) Market availability of programming skills and experiences

There should be sufficient availability in the market for programmers with the right skills and experiences. This is one of the reasons that older programming languages used in the mainframe environment have to be migrated as the number of programmers with the appropriate skills and experiences have declined significantly.

3.5.2.2 Coding Standards

There is no design consideration for this component. However, refer to Best Practices for related information.

3.5.2.3 Error Handling Standards

There is no design consideration for this component. A good application framework, as described above, can aid the reusability of error handling standards. A set of standard error handling codes can be defined and invoked depending on the error type. Please refer to Best Practices for related information.

3.5.3 Application Testing

Application testing is necessary to ensure applications performed to expectations. Typically, testing would be carried out at the end of a logical completion of task or phase such as unit test, module test and user acceptance test (UAT). There are also different methods of testing such as white-box test versus black-box test, and manual test versus automated test. The relevant type and method of testing is dependent on the nature of the application. Instead of describing these test types and test methods, design consideration focuses on the approaches to testing.

Areas of consideration for application testing:

(a) Customise a testing methodology

Regardless if the government agency has in-house application development team or outsourced its application development, it has to customise a testing methodology depending on the nature of its applications. A testing methodology would define the testing type, and testing method required for the government agency

(b) Carry out static testing before dynamic testing

Static test focuses testing the form and structure of the applications while dynamic test looks at the application behaviour. Static testing helps in the early detection of defects which are often cheaper to remove than those detected during dynamic testing. Examples of static testing are document reviews, desk checking, functional test and unit test. Examples of dynamic testing are load and stress tests

(c) Use of automated testing tools

If the tests are going to be run repeated several times, considering using automated testing tools where possible. Even though automated testing tools may require some developmental effort, the testing cycle is much shorter as compared to manual testing

(d) Carry out load / stress testing where possible

Load or stress testing will provide an accurate estimate on the number of concurrent users supported and help to rectify any performance bottleneck prior to implementation. Load / Stress testing is recommended for public-facing applications (i.e. accessible by public from the Internet) and large, complex applications

(e) Carry out security testing

If the application is public-facing, consider performing security testing prior to going live. This provides opportunities for the development team to avoid any kind of exploitation by hacking.

3.5.4 Application Deployment

Application deployment, often overlooked as an unimportant activity, ensures the application success when implemented in the live environment. There are many activities needed to transfer the application from test to live environment.

Areas of consideration when deploying applications:

(a) Directory structure

Consider creating a web application directory structure which identifies the various types of files that would be used by the application and therefore the corresponding rights on the web server

(b) Version tracking

Versioning guidelines should be defined and used so that it is easier to manage the various version of the application in the production, staging, QA/test and development environment

(c) Deployment checklist

Create a standard deployment checklist to ensure that the basic activities are carried out completely and in the right sequence. Where possible, have a reviewer to ensure that the deployment activities are checked according to the checklist.

3.5.5 Application Configuration Management

There is currently no architecture design consideration.

3.6 Standards Classification

Each technical standard in the different domains is identified with a classification. Table SA-5 describes the definition of these two standards classifications.

Standards Classification / IT Systems	Mandatory*	Recommended for Enhancements and New IT Systems*
	This is the minimum technology standard that is mandatory for existing, enhancements and new IT Systems. Government agencies shall migrate to this mandatory classification.	This is the technology standard that supplements the Mandatory standard. This standard is applicable to enhancements to existing IT Systems and all new IT Systems. It is advisable for the government agencies to adopt this standard where possible.
Existing IT Systems	✓	✗
Enhancements to existing IT Systems	✓	✓
New IT Systems	✓	✓

***** Government agencies are to seek ITA advice for exemptions.

Table SA-5: Standards Classification

3.7 Technical Standards and General Standards

Please refer to Appendix SA-1 for the list of technical and general standards.

3.8 Best Practices

3.8.1 Development Methodology

Best practices for development methodology include:

(a) Adopt a development methodology

Depending on the application requirements and complexity, the waterfall or iterative or a combination of the two methodologies can be used to design and develop the application. Some of the best practices are:

(i) Establish a standard methodology for the government agency

The use of a standard methodology would ensure a consistent application development practices. Government agencies can use either the waterfall, iterative, or combine and customise the two methodologies for agency specific use. The standard methodology can be used for both in-house and outsourced application development.

(ii) Define clear deliverables

A good development methodology would define clear deliverables at each stage or phase. Examples of clear deliverables include approved project plan, documented & signed user requirements specifications, approved application architecture & design, approved test & implementation plans, approved user acceptance test & system test reports, and reviewed application documentation.

(b) Technique for use case modelling

The following are some techniques for better Unified Modelling Language (UML) use case models:

(i) Write from the point of view of the actor and in the active voice

Use cases should be written in the active voice instead of the passive voice. In addition, each use case should be written from

the point of view of the actor. After all, the purpose of the use cases is to understand how your users will work with your system.

(ii) Write scenario text, not functional requirements

The use case should describe a series of actions that provide value to an actor and not a collection of features. For example, an “Enrol in Seminar” use case will describe how an employee interacts with the system to sign up for a seminar. It should not describe what the user interface looks like, or how it works. This important information should be described in the user interface model. Object oriented analysis is complex, which is why there are several models to work with, and we should apply each model appropriately.

(iii) Create a use case template

Use cases include a fair amount of information that should be documented for ease of references. Agencies should consider developing their own templates for the documentation. A sample is as follows:

Name	The name should implicitly express the user’s intent or purpose of the use case.
Identifier [optional]	A unique identifier that can be used in other project artefacts to refer to the use case.
Description	Summary of the use case.
Actors	The list of actors associated with the use case. Although this information is contained in the use case itself, it helps to increase the understanding of the use case when the diagram is unavailable.
Status [optional]	An indication of the status of the use case, typically one of work in progress, ready for review, passed review or failed review.

Frequency	How often this use case is invoked by the actor. This is often a free-form answer such as once per each user login or once per month.
Preconditions	A list of the conditions, if any that must be met before a use case may be invoked.
Post conditions	A list of the conditions, if any that will be true after the use case finishes successfully.
Extended use case [optional]	The use case that this use case extends (if any).
Included use case [optional]	A list of the use case this use case includes.
Assumptions [optional]	Any important assumptions about the domain that has been made when writing the use case. At some point, these assumptions should be verified and evolved either into decisions or into parts of the basic course or alternate courses of actions.
Basic course of action	The main path of logic an actor follows through a use case. Often referred to as the happy path or the main path because it describes how the use case works when everything works as it normally should.
Alternate courses of action	The infrequently used paths of logic in a use case, paths that are the result of an alternate way to work, an exception, or an error condition.
Change History [optional]	Details about when the use case was modified, why and by whom.
Issues [optional]	A list of issues, if any that is related to the development of this use case.

Decisions	A list of critical decisions pertaining to the content of the use case. It is important to record these decisions to maintain a "group memory".

Table SA-6: Use Case Template

(iv) Organise the use case diagrams consistently for ease of readings

The common practice is to draw inheritance and extend associations vertically, with the inheriting/extending use case drawn below the parent/base use case. Similarly, include associations that are typically drawn horizontally. These are simple rules of thumb which, when followed consistently, result in diagrams that are easier to read.

(v) Document use cases to aid the development of a user manual

The purpose of a user manual is to describe how a user can work with the system. Since each use case describes a series of actions taken by actors using the system, the same information can be used to develop the user manual.

(vi) Use case versus steps within a use case

After developing the use cases, use case developers should review and consider if the use case adequately represents the key goal of the application from the user's perspective. Quite often use cases defined with names like "Validate customer number" is more of a step in a use case rather than a use case itself. In this example, the use case is likely to be "Sales Order" while the step within this use case is "Validate customer number".

(c) Guidelines for documenting use case models

For clarity, the use case model should be documented using the following guidelines. Other than making the model easier to understand

and maintain, the models will improve communication internally within the team and externally with partners and customers, thereby reducing the chance of costly misunderstandings.

All diagrams comprise three basic elements: bubbles, lines and labels. As a result, a common set of documentation guidelines applies to most diagrams:

(i) Avoid crossing lines. Two crossed lines on a diagram can easily be misread. If we cannot avoid crossing lines, draw one so that it "hops over" the other to make their difference explicit

(ii) Avoid diagonal or curved lines. Straight lines, drawn either vertically or horizontally, are easier to follow visually. Placing bubbles on diagrams as if they are centred on a graph's grid point (a built-in feature of many software-based modelling tools) makes it easier to connect the bubbles with only horizontal and vertical lines

(iii) Draw bubbles of consistent size. The larger a bubble appears, the more important it seems to be. Unless you want that effect, make the bubbles of uniform size. With some modelling tools, however, bubbles may resize automatically based on their contents, so this may be beyond our control

(iv) Include white space. White space is the empty area between modelling elements on our diagrams. When bubbles crowd one another, it can be difficult to distinguish which labels go with which bubbles and lines, decreasing the diagram’s readability

(v) Organise diagrams from left to right, top to bottom. If there is a starting point for reading a diagram, such as the initial state of a state machine diagram or the beginning of the flow of logic on a

sequence diagram, place it towards the top left corner of the diagram and continue from there

(vi) Show only what you have to. Diagrams that have too many details are difficult to read because they are too dense. Include only critical information on the diagrams and not include anything extraneous

(vii) Keep your diagrams small. It is often better to have several diagrams showing various degrees of details than one complex diagram that reveals everything. A good rule of thumb is that a diagram should have five to nine bubbles, because there is a limit on the amount of information that we can deal with at once.

Set and follow naming conventions. This ensures consistency within our models and hence increases their readability. Diagrams should use consistent and recognisable domain terminology. This is particularly true for domain-oriented diagrams that our project stakeholders are likely to be involved with.

3.8.2 Application Design

Best practices for application design include:

(a) Government agencies should have an application framework for in-house development as the framework could:
  (i) provide a controlled environment for developing applications and interfacing with other systems, hence ensuring quality of work
  (ii) abstract the complexities of the infrastructure away from the logic of the application. Hence enable developers to focus on implementing business logic. This can lead to larger productivity gains and a more maintainable application

(b) Government agencies should also specify that outsource application vendors to use a framework to build applications. It ensures quality applications that are structured and easier to maintain.

(c) Adopt N-tier architectures and design principles for distributed applications

While mainframe and traditional 2-tier client/server system still have their important role in serving the business needs of the various agencies, N-tier architectures and application design principles should be adopted for distributed applications. This will help to enforce logical separation of functionality, such as keeping the data, business logic and presentation layers separate from each other. At the same time, the application is able to scale and change at each layer with minimum impact on the other layers as requirements and technology change, thus providing improved scalability, manageability, and resource utilisation. In addition, existing non N-tier applications may be able to participate in distributed architectures through the use of suitable middleware technologies in the integration tier. Application design should consider exposing functionalities as reusable services. This will enhance interoperability and reduce inter-dependency with external systems.

The table provides a guide for making decision on the application tier. The factors may not cover all areas of consideration as every project is unique in nature and there is no one size fit all solution. However, given the factors below, one is able to quickly make some initial assessment and decision on the tier for the intended project.

Decision Factor for Tier Approach	1-Tier	2-Tier	3-Tier	N-Tier
Number of Users				
Small (> 1 and < 20)	*	*	*	
Medium (>20 and < 99)		*	*	*
Lager (> 100)			*	*
Domain				
Intranet	*	*	*	*
Internet			*	*

Decision Factor for Tier Approach	1-Tier	2-Tier	3-Tier	N-Tier
Processing Requirements - utilisation of the CPU resource to perform business logic and validation.				
Low	*	*	*	*
High			*	*
Security - logging mechanisms, monitoring devices, as well as Appropriate Authentication, ensuring that the device and system is always secure.				
Low	*	*	*	
High			*	*
Performance - capability of a system to provide a certain response time.				
Low	*	*	*	*
High			*	*
Vertical / Horizontal Scaling - characteristic of a system to increase performance by adding additional resources. Vertical: add more hardware resources to the same machine, generally by adding more processors and memory. Horizontal: add more machines to improve performance.				
Vertical		*	*	*
Horizontal			*	*
Web Based Application Required?				
No	*	*		
Yes			*	*
Any integration with External system or centralised Services?				
No	*	*		
Yes			*	*
Availability of System - ensures a certain degree of operational continuity				
Low	*	*	*	*
High			*	*
Need to support multiple User Interface?				
No	*	*	*	*
Yes				*
Need for instant availability of application updates across the entire organization?				
No	*	*	*	*
Yes				*
Need for rich communication such as connectionless messaging, queued delivery, publish-and-subscribe, and broadcast.				
No	*	*	*	*
Yes			*	*

Table SA-7: Guide to Application Tier

(d) Base design decision on what works now, not a promise of functionality in the future. This is especially true for J2EE applications, where

specifications are released before production-quality implementations. Also vendors' implementations may vary. Hence specified features may not always prove to work in the real world.

(e) Have experienced architects or designers involved in the design process, especially when implementing an enterprise application. These experts can help to mentor less experienced developers in the process, hence building a deeper reserve of expertise.

(f) The application design should take into account usability. There is a common look for the Oman eGovernment Services Portal. Government agencies should attempt to follow the common look and feel standard for all application user interfaces (refer to TRM Service Access Domain for specific guidelines).

3.8.3 Application Development

Best practices for application development include:

(a) Adopt a coding standard for naming convention, code documenting convention, programming convention and code formatting convention, All these conventions will make the code more readable across development teams, improve the quality, performance and efficiency of the codes.

(b) Adopt an error logging standard to aid in determining the cause of an error in code. Some programming languages provide standard mechanisms for implementing error logs such as Java Exception handling which the development team can leverage on. Errors should be classified into severity levels such as "warning," "input error," and "fatal error", to facilitate developer's response. There are application frameworks that allow error logging standards to be defined and implemented (refer to Section 2.7.2 Application Design).

(c) Write effective error messages

Error messages should describe the problem in plain simple terms as users do not have the same level of technical knowledge to be able to interpret messages written in technical terms. The message should be worded to help the user to understand the problem and provide information on what the user can do to moving ahead. The developer should build in flexibility to allow modifications of error messages without changes in the source code. Some of the points to note are:

(i) Do not use exclamation point either in the error message or as an icon

(ii) Do not display the error messages in capital letters

(iii) Do not use the word 'Error' as it has strong negative connotations

(iv) Phrase the message as a question that can be answered with a simple 'Yes' or 'No' to reduce ambiguity.

(d) Document the source code

A simple, commented program will have a comment for every other line of code that needs explaining. Code documentation is beneficial to developers - it segregates portions of the code, makes it more readable as well as easier for further modifications. Well-documented code is maintainable and allows anyone to continue working on the same piece of program after a period of time. The Java programming language has a feature that allows HTML API documentation to be directly generated from code documentation. The Visual .Net IDE from Microsoft also has this feature for all the .Net languages.

(e) Program Header

From the program header, the developer gets a quick overview on what the program is supposed to do. It is important the developer who may be debugging the code to be in the right mindset so that he or she can understand why the program was coded in a certain way. The

description should be short, yet detailed. After the description comes the name of the creator and then the date. Some Version Control software may also be incorporated to update the program version number in the program header.

After that there may be an optional section for notes - any oddities that may have thrown in at the last minute or any incomplete areas of the program (e.g. to-do list). Other things to include may be the names of the dependencies programs. Note that sensitive information about the program should not be listed in the header such as database id and password.

(f) Method/Function Header

The method/function (depending on whether you are using object-oriented or procedural languages) header lets the reader know exactly what the method/function does, and why it does it. There is a brief description, and a variable list. The variables section describes what each variable is supposed to do or represent. Like in the program header, an optional notes section, may describe any exceptional qualities that the method/function may encounter.

(g) Refrain from over commenting

The next step is to go through the code and add in additional comments to explain certain areas in more detail. The purpose of comments is to make code easier to read. However, if there are too many unnecessary comments and if these comments are not placed correctly, it will have a negative effect by making the code harder to read because you can't tell the difference between the code and the comments.

(h) Descriptive Variable Names

There is a way to decrease the developer's commenting workload - using descriptive variable names. Variable names should be short and yet self-explanatory.

The same rule applies to any type of name – method/function, structure, class, etc. As long as the variable name can describe what it is supposed to do, developers do not have to describe it themselves.

3.8.4 Application Testing

Best practices for application testing include:

(a) Use automated test tools

Manual testing can be quite overwhelming. The use of automated test tools can reduce testing effort. If test data cases have been catalogued and preserved, test effort will be minimised since less time will be spent on re-creating such test cases. Automated tools require effort to prepare the test scripts and the assumption that a script can be re-use may not be always true. Selection of the appropriate tool is important as each tool has its own strength/merits.

With the trend towards outsourcing, ability to ensure code quality has become a challenge. With newer generation of test tools which has the capability of analysing code complexity and maintainability, it is possible to determine potential code quality issue. It is advisable for project team to require their software developer to make use of such tools for conducting periodic review as evidence of quality assurance.

With multi-tier application, server and network environments, problem and performance troubleshooting to ensure the Service Level Agreement (SLA) of applications is becoming increasingly tricky. To address this, there is an increasing need to consider application specific performance monitoring and tuning tools. These tools may be used during pre-production for application assurance or could be deployed into production for performance management and diagnostics.

(b) Performance Usability Testing

For Web-based application solutions, investment should be made in usability testing to ensure delivered products are user-friendly. Usability testing also reduces overall development costs by enhancing requirements understanding and catching problems early in the development cycle.

Before conducting a usability testing, factors to consider include:

(i) Determine the user profile/target audience;

(ii) Preparing multiple scenarios or design so as to ensure sufficient user feedback;

(iii) Be prepared for all potential feedback including those, which may be inappropriate, controversial and culturally unacceptable.

The basic elements of a usability test are quite simple and involve three participants: facilitator, observer, and user. Prior to running a test, the facilitator should perform a dry run of the test to ensure the scenarios and equipment is ready. During a test, the observer records the events while the test facilitator introduces the tasks to the user. The goal is for the user to complete a set of tasks in a set amount of time. This will enable the test to produce results proving cost/time savings. Usability testing should also include rapid prototyping sessions in which users and developers work together. Usability testers may refer to ISO / IEC for recommended standard method for reporting usability test findings.

In addition to pre-production usability testing, it is also important to continually monitor and gather feedback from users after implementation. Other than using surveys, click-logs and traffic logs that track functional popularity, there are also an emerging start-up market offering tools or services will track the user's actual click locations and actions. These offerings in addition to tracking functional popularity, also generates click density maps that help webmasters evaluate if they are

successfully directing their user's attention areas and usage preferences.

(c) Application Security Testing

Security review, penetration testing, tools and basic testing practices can assist in uncovering known vulnerabilities that are typically found in the Web application. In order to ensure complete application security, it is recommended that the following application security testing covers areas such as authentication, authorisation, information leakage, field variable, session time-out and log-out, cache control, server side application logic and client-side software vulnerabilities. For security related information, please refer to Oman National Computer Emergency Readiness Team.

(d) Interoperability Testing

Other than the usual unit, functional, regression and load testing, web service development has promoted an area of testing that is important but often subsumed under other integration requirements – Interoperability testing.

To ensure better integration of systems through Web Services, it is recommended that interoperability testing is carried out using Web Services Interoperability (WS-I) Basic Profile. WS-I Testing Tools are available and are designed to help developers determine whether their Web services are conformant with the WS-I profile guidelines.

Table SA-8 summarised a recommended set of minimum testing to be performed by the various application types.

Type of Application \ Recommended Type of Testing	Client / Server Application	Web-enabled Application	Interface & Integration Component
Manual Functionality Testing	*		*
Automated Functionality Testing		*	
Automated Regression Testing		*	
Automated Load Testing		*	
Automated Performance Testing	*	*	*
Usability Testing	*	*	
Security Testing		*	
Interoperability Testing			*

Table SA-8: Minimum Testing by Application Types

3.8.5 Application Configuration Management

Best practices for application configuration management include:

(a) Based on configuration management best practices (ANSI/EIA-649), four principles were identified for an effective configuration management. The four principles are:

(i) Configuration Identification – basis from which the configuration of products are defined. The configuration documentation defines the performance, functional and physical attributes of the application and system infrastructure.

(ii) Configuration Control – changes to the application are accomplished using a systematic and measurable change process. Changes to the configurable items are controlled and coordinated after approval is given after the change control process.

(iii) Configuration Accounting – track each version of release and record all changes made to previous baseline to reach new baseline.

(iv) Configuration Verification – ensure new baseline has all planned and approved changes incorporated, and would be done before product release.

(b) Establish a Change Control Board to review, control, and approve all changes in a software application.

(c) Develop an overall Configuration Management Plan (CM Plan). The CM Plan will describes the policy, responsibilities, schedules and procedures for a uniform adoption and implementation methods for the projects. The procedures of the CM Plan (e.g. approval process, tracking of changes, etc.) should be documented in a separate document for better maintainability and audience focus.

3.9 Obsolete Technologies

Government agencies have to ensure that they do not have any obsolete technologies in this domain. Please ensure compliance by referring to OeGAF Obsolete Technologies Compliance List.

4. Service Access Domain

4.1 Intent

The Service Access Domain defines the technology categories, technology components and associated standards for access channels that allow users to interact with the requested applications and for the applications to communicate responses to the users. It highlights key architecture design considerations and recommends best practices for service access implementation.

4.2 Domain Design Principles

When designing a service access solution, the following design principles should be observed:

Principle 1: Leverage on the Internet and Intranet

The Internet and Intranet, as an information and service delivery medium, will be the first technology option considered for implementing new applications and performing a rewrite or significant maintenance on legacy application. By leveraging on the Internet and Intranet, it allows the user to access information anytime or anywhere. It also provides a consistent presentation method, reduces total cost of ownership and reduces cost of service delivery.

Principle 2: A Variety of Delivery Mechanisms and Interfaces to Deliver Electronic Government Services to a Wide Range of Devices

The delivery mechanism includes electronic mail, multimedia web pages, streaming audio or video, voice response units, and wireless transmissions. The terminal devices include, but are not restricted to, voice telephones, personal computers, cellular telephones, personal digital assistants, and kiosks.

Principle 3: Design to Support Users with Limited Physical Abilities

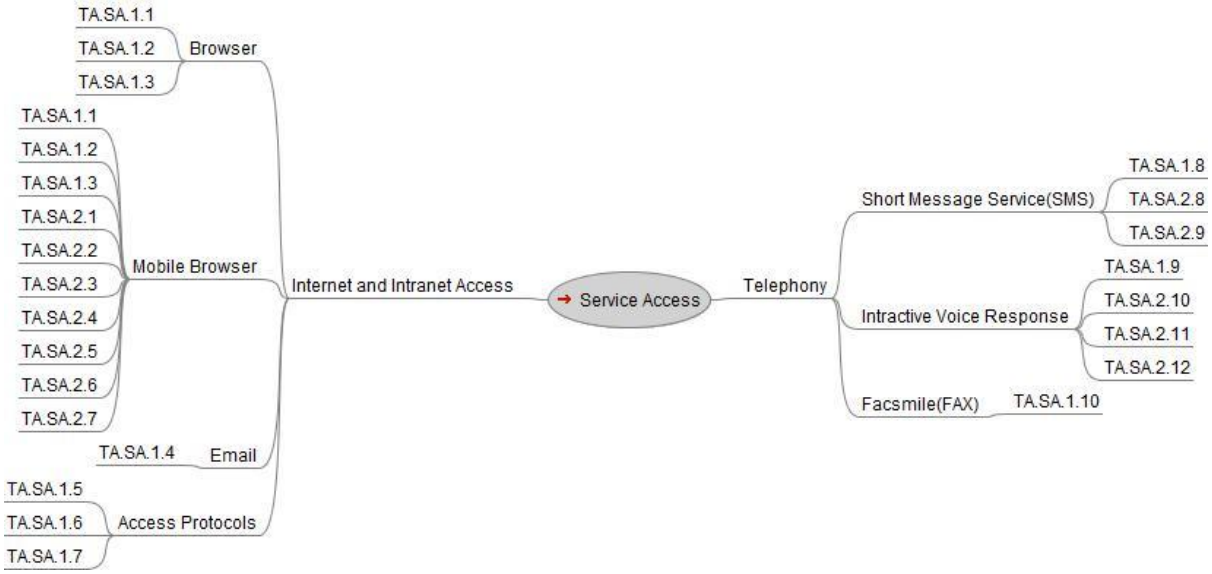
Electronic government services must be ability independent and includes delivery to users with limited physical abilities. This includes the ability to listen to the subject content (in Arabic or English) where applicable.

Principle 4: Plan Service Access for High Availability

The service access must be planned with high availability and caters for round-the-clock service availability because the users should be able to access the services from any part of the world.

4.3 Technology Categories and Technology Components

Figure SA-10 shows the association between the technology categories, technology components and its relevant standards.



(need to replace diagram above)

Figure SA-10: Mapping of Categories, Components and Standards for Service Access Domain

Table SA-9 describes both the technology categories and technology components of the Service Access Domain.

Technology Categories	Technology Components
Web Application Platform	**Web Server** A Web Server, also known as an HTTP Server, is a computer connected to a TCP/IP based network that runs software implementing the server side of the HTTP protocol, thus allowing it to receive and respond to HTTP requests from a browser. The computer hardware, operating system, and Web Server software, taken together, constitute the Web Server.

Technology Categories	Technology Components
	Web Proxy Server A Web Proxy Server is a server that acts as an intermediary for requests from clients seeking resources from other servers. The proxy server evaluates the request according to its filtering rules.
	Portal Server A portal server allows an organisation to unify its various web sites or portals (customer portal, business partner/supplier portal, employee portal, etc.) into one single portal so as to promote information sharing, collaboration and interaction among its users by presenting content in a focused and personalised manner. A portal server typically aggregates the content of the various organisations' web sites and offers common content manipulation services such as content publishing, content sharing, content search, personalisation, collaboration and internationalisation/ localisation of content to its users. It also typically comes with a wide range of mechanism to facilitate integration of the portal with the various disparate content sources.

Technology Categories	Technology Components
	To an end user, a portal is a single point of access to an organisation's published content and computing resources.
	Application Server An application server is a software framework dedicated to the efficient execution of procedures for supporting the construction of applications. The term was created in the context of web applications. In these, the application server acts as a set of components accessible to the software developer through an application programming interface (API) defined by the platform itself.
	Integration Server Integration Servers provide integration services for distributed transaction across disparate application systems. It also includes distributed system middleware features like message queue and Transaction Processing (TP) monitor.
	Database Server A database server is a computer program that provides database services to other computer programs or computers, as defined by the client-server model. The term may also refer to a computer dedicated to running such a program. Database management systems

Technology Categories	Technology Components
	frequently provide database server functionality, and some DBMS rely exclusively on the client-server model for database access. Such a server is accessed either through a "front end" running on the user's computer which displays requested data or the back end which runs on the server and handles tasks such as data analysis and storage.
	Directory Service A directory service is the software system that stores, organises and provides access to information in a directory. It allows the look up of values given a name.
	Search Engine A search engine is a software program that searches for sites based on the words that are designated as search terms. Search engines look through the databases of information in order to find what it is that you are looking for.
	Integration Broker An integration broker enables applications to exchange information in dissimilar forms by handling the processing required for the information to arrive in the right place and in the correct format.
	Transaction Processing Monitors

Technology Categories	Technology Components
	Transaction Processing (TP) Monitors manage transactions both within and across systems. They handle housekeeping tasks such as rollback and resource co-ordination.
Internet and Intranet Access This refers to the technology that enables platform independent, intra and inter-agency business communications as well as communications with external business partners and customers.	**Browser** This is a common front-end component for retrieving, presenting and traversing information resources on the World Wide Web. There are now a number of widely used browsers.
	Mobile Browser (also known as Micro Browser and Mini Browser) This is a type of browser designed to be used on the small screens of many different types of mobile devices. It is commonly used on mobile phones that require a gateway to translate Web pages, news feeds and other Internet content.
	Access Protocols This refers to all Internet/Intranet access enabling protocols and methods that fall into the realm of process-to-process communications via an Internet Protocol (IP) network.
Telephony This refers to technology components that facilitate seamless transmission of information between applications and users via telephony related communication channels.	**Interactive Voice Response (IVR)** IVR is an interactive technology that can recognise voice and keypad inputs from phone systems. In addition, it can respond to inputs via pre-recorded voice messages or dynamically generated audio.

Technology Categories	Technology Components
	Facsimile (Fax) A fax is a digitised image of text and/or pictures, represented as a series of dots (bit map). Faxes can be sent and received through telecommunication channels.
Collaboration Management	**Electronic Mail (Email)** Email is a method of exchanging digital messages. E-mail systems are based on a store-and-forward model in which email computer server systems accept, forward, deliver and store messages on behalf of users, who only need to connect to the email infrastructure, typically an email server, with a network-enabled device (e.g. a personal computer) for the duration of message submission or retrieval.
	Instant Messaging (IM) Instant messaging (IM) is a form of real-time communication between two or more people based on typed text. The text is conveyed via devices connected over a network such as the Internet.
	Short Message Service (SMS) SMS is a communication service standardized in the mobile communication system, using standardised communications protocols allowing the interchange of short text messages between mobile telephone devices.

Technology Categories	Technology Components
	Collaboration Workspace A collaborative workspace or shared workspace is an inter-connected environment in which all the participants in dispersed locations can access and interact with each other just as inside a single entity. The environment may be supported by electronic communications and groupware which enable participants to overcome space and time differentials. These are typically enabled by a shared meta model, common information, and a shared understanding by all of the participants regardless of physical location.
	Video Conferencing This is a set of interactive telecommunication technologies which allow two or more locations to interact via two-way video and audio transmissions simultaneously.
	Enterprise Content Management (ECM) ECM refers to the technologies, tools, and methods used to capture, manage, store, preserve, and deliver content across an enterprise. At the most basic level, ECM tools and strategies allow the management of an organisation's

Technology Categories	Technology Components
	unstructured information, wherever that information exists.

Table SA-9: Service Access Technology Categories and Components

4.4 Architecture Design Considerations

4.4.1 Web Application Platform

The design of the web application platform has to primarily support the tier architecture (i.e. 1-tier, 2-tier, 3-tier or N-tier application type).

4.4.1.1 Web Server

The function of the web server is to receive and reply to client request over the Internet / Intranet, typically via HTTP.

Areas of consideration when selecting web server:

(a) Support for open standards.

The web server should support open standards such as HTTP, HTTPS and TCP/IP. It should preferably be supported by the various common operating systems.

(b) Performance of web servers.

Web servers should incorporate features for scalable performance such as load-balancing, clustering and failover.

4.4.1.2 Web Proxy Server

The function of the web proxy server is to alleviate the load on the content servers and the Internet.

Areas of consideration when selecting proxy server:

(a) Purpose of proxy server

Consider the purpose of the proxy servers and hence the types of proxy server. A description of some of the server types are listed as follows:

(i) Forward Proxy Servers

These proxy servers are located at the edge of the Internet Service Provider's (or enterprise's) network and retrieves web requests on behalf of the users. This saves excessive Internet traffic for pages that are commonly accessed.

(ii) Reverse Proxies

The servers are placed at the service provider's data centre and will attempt to serve the content to the users requesting for it, so that the origin servers are not bombarded by requests.

(iii) Transparent Proxy Cache

Transparent proxy cache is similar to a forward proxy server. It transparently serves its users without the users having to specify the usage of a proxy server. The traffic is automatically diverted to the transparent proxy, usually via a Layer 4 switch.

(b) Performance of proxy servers.

If performance is required for rapid lookup (especially in a clustered configurations), consider hardware-based proxy rather than software-based proxy server instead.

(c) Cost of proxy server.

Consider software-based proxy server if cost is a concern.

4.4.1.3 Portal Server

The function of the portal server is to unify the web servers as the main access point to the application. It receives all requests from the clients and sends them for processing. Upon receiving the processed information, the portal server returns the results back to the clients. A portal server aids information sharing and allows the customisation of the information presented to clients.

Areas of consideration when selecting portal server:

(a) Support for open standards

Portal framework should support open standards for connectivity and compatibility e.g. provide support for Light-weight Data Access Protocol (LDAP) and common protocols for information exchange built on XML.

(b) Integration with other applications

The portal servers should support seamless integration with other essential enterprise applications. Actual integration requirements for a portal server would vary depending on agency's requirements. The following provides some considerations for a portal product's integration functions:

(i) Availability of portal adaptors to connect to directory servers, databases, file systems, business intelligence tools, content and document management systems, groupware and office productivity tools (e.g. e-mail, word processing documents, spreadsheets) or applications (front-office, back-office and e-business systems). Note that it is not the number of adaptors provided that matter, but the degree of integration provided by the adaptor

(ii) Availability of support for out of the box portlet integration with common systems, such as Enterprise Resource Planning (ERP), databases, etc

(iii) Availability and ease of integration with best-of-breed technology components from other vendors (especially those not included in the basic portal offering). As far as possible, the use of a standard based approach to integration (e.g. J2EE Connector Architecture, XML Web Services, XML) is preferred rather than the proprietary method of connecting to backend systems.

(c) Availability of Open APIs

Portal servers should include open APIs (or portal development kits) to allow for the integration of external application systems (e.g. agency's own proprietary applications) that may not be able to integrate with the portal out of the box. Portal should also support various programming and technology interfaces as needed to access the functions to be integrated into the portal (e.g. EJB, COM+, CORBA/IIOP, message queuing, Java, C++, etc).

(d) Administration tools

Portal servers should have sufficiently rich administration tools to support the efficient maintenance of a portal. The following should be included: remote administration, graphical user interface (GUI) administration tools and delegated administration facilities.

(e) Performance

Portal servers should incorporate features for ensuring robust and scalable performance such as load-balancing, clustering, fault-tolerance, failover, connection pooling, caching and other performance and scalability features. While portal servers that are designed to run on an application server normally inherit the application server's capabilities in these areas, standalone portal solutions must implement such services as part of their products to be considered robust and scalable solutions.

(f) Delivery channels support

Availability of support for a broad range of delivery channels (PDAs, mobile phones, interactive digital TVs).

(g) Out-of-the-box functionalities

Ability to provide basic out-of-the-box functionalities such as templates, workflows and service desk.

(h) Portlets reusability

Ability to reuse portlets developed by a particular portal product across agencies.

4.4.1.4 Application Server

Application server provides services and components for easy rapid development and deployment of enterprise application systems. Application Servers can be categorised as follows:

(a) Standard Application Servers

Support the features provided by Web Servers e.g. Java Server Pages (JSP), Active Server Pages (ASP) or HTML pages. In addition, they may support technology components such as Enterprise Java Beans (EJB), J2EE Connector Architecture (JCA), and Java Messaging Service (JMS).

(b) Enterprise Application Servers

Support the features provided by Application Servers and includes clustering solutions (e.g. failover capabilities, replication and load balancing). These servers are intended for production use at an enterprise level. Descriptions of these clustering solutions are provided in the following Table SA-10:

Component/Services	Description
Load Balancing	Ability to allocate resources for tasks to achieve best response time based on load factor
Failover	Automatically reconnects all clients to a backup machine
Security	Provides Authentication, Authorisation and Audit Trail

Component/Services	Description
Administration	Monitoring, application configuration, and user/application administration
Container API	J2EE container or equivalent implementation
Metadata management	Contains configuration information on logical and physical resources, application rules, security information, etc. in a directory/database

Table SA-10: Enterprise Application Servers Clustering Solution

Generally, when upgrading from an Application Server to an Enterprise Application Server (of the same product), sharing the same codes is possible and practical. The application will run as it is, without modifications, by replacing the server license. However, for the application to leverage on the clustering solutions provided by the Enterprise Application Server, modifications to the application are required.

Areas of consideration when selecting application server:

(a) Review application needs

The government agency must carefully analyse the current and future applications' needs before selecting an application server to ensure that the selected product can support additional functionalities that will be needed in the near future, and the investment in developing and deploying the application server is protected.

(b) Application server version

Most application servers have different versions like standard and enterprise as mentioned above. Select a version that meets the application requirement.

(c) Product certification

The application server should preferably be certified by the product vendor.

Microsoft .NET versus J2EE Application Server

Two major application servers are Microsoft .Net and J2EE. Table SA-11 documents the key differentiators between them. Government agency can refer to this table when considering which application server to adopt for an application.

Key Differentiators	Microsoft .NET	J2EE
Portability	The .NET core works on Windows only though theoretically it supports development in many languages (once sub-/supersets of these languages have been defined and Intermediate Language (IL) compilers have been created for them). Its SOAP capabilities will allow components on other platforms to exchange data messages with .NET components. While a few of the elements in .NET, such as SOAP and its discovery and lookup protocols, are provided as public specifications, the core components of the framework, IL runtime environment, ASP+ internals, Win Forms and Web Forms component "contracts", etc. are kept by Microsoft, and	J2EE works on any platform with a compliant Java VM and a compliant set of required platform services (e.g. EJB container, JMS services). All of the specifications that define the J2EE platform are published and reviewed publicly, and numerous vendors offer compliant products and development environments. But J2EE is a single-language platform. Calls from/to objects in other languages are possible through CORBA, but CORBA support is not a ubiquitous part of the platform.

Key Differentiators	Microsoft .NET	J2EE
	Microsoft will be the only provider of complete .NET development and runtime environments. There has already been some pressure by the development community for Microsoft to open up these specifications, but this would be counter to Microsoft's standard practices.	
Language Support	.NET is language independent and can use any language once a mapping exists from that language to IL. Several third-party vendors have produced language compilers that target the Common Language Runtime (CLR); examples include NetCOBOL from Fujitsu, and Visual Perl and Visual Python from ActiveState. Because all .NET languages share a common type system (CTS), developers can safely pass types written in one language to code written in another. They can also use the unified Framework Class Libraries in any .NET language, saving them from having to learn how to work	Tied to Java. To use another language, interface technology like Java Native Interface (JNI) or Web Services will need to be used.

Key Differentiators	Microsoft .NET	J2EE
	with many different implementations.	
Tools Support	Microsoft's Visual Studio .NET provides a robust IDE for building .NET applications which include Windows applications, Web applications, or XML Web services. From a developer's viewpoint, applications are built using a single IDE though different .NET languages can be chosen. The Visual Studio .NET Server Explorer allows developers to access server resources (e.g. message queues, performance counters, data sources) without leaving Visual Studio .NET.	Java features a host of tools, which provides the developer with choices, though it may present difficulty in choosing the right tool for a given job. More generally, debugging support is good via the Java Platform Debugging Architecture (JPDA), and the arrival of Ant and JUnit have given the Java community a standard build tool and unit testing framework, respectively. There are a number of freeware and shareware available for Java IDE.
User Authentication and Authorisation	.NET suffers from tight integration with IIS, without which it is not really capable of performing authentication. In terms of access control, it does provide a convenient mechanism that meshes nicely with its Code Access Security (CAS) features.	Java, in addition to the standard authentication types, offers the powerful Java Authentication and Authorization Service (JAAS) mechanism as its primary vehicle for adding authentication and Principal-based authorisation to Java applications, which adds a lot of flexibility to design choices.

Key Differentiators	Microsoft .NET	J2EE
Security Features	Microsoft products have done best in the closed, homogeneous environment of all-Windows networks. This type of environment allows for system integration and utilisation of .NET security features to their fullest potential.	Java performs well in heterogeneous environments. In the case of a mixed environment, Java's platform-independent security features may be more useful than those of .NET.

Table SA-11: Comparison of .NET and J2EE Application Servers

4.4.1.5 Integration Server

There is no design consideration for this component.

4.4.1.6 Database Server

There is no design consideration for this component. Refer to IRM for design considerations.

4.4.1.7 Directory Service

A directory is an information source used to store information about objects such as users, applications, and network resources, in an hierarchical tree format that can be set up to represent an organisation chart.

Areas of consideration for using directory service (versus database):

(a) Frequency of read/search versus updates

Directory service is optimised for read access. Hence they should be used when reads/searches occur more often than updates (for example telephone directory). If there is frequent need to update to store dynamic information, database server should be used instead.

(b) Access Need

If there is a need to read data for information or authentication, then, consider using directory service instead of database server.

4.4.1.8 Search Engine

A search engine enables a user to find information based on his/her own search criteria, rather than using predefined paths. It is used to quickly find out where information is located. It presents a common interface to all sources of information on Web servers. It can be offered either as a service (e.g. Yahoo or Google) or as a product, either hardware-based or software-based. Figure SA-11 illustrates the components that make up a universal search engine.

An enterprise search engine is capable of searching against multiple content repositories and databases, as well as providing advanced search features such as concept-based search, content categorisation and personalisation.

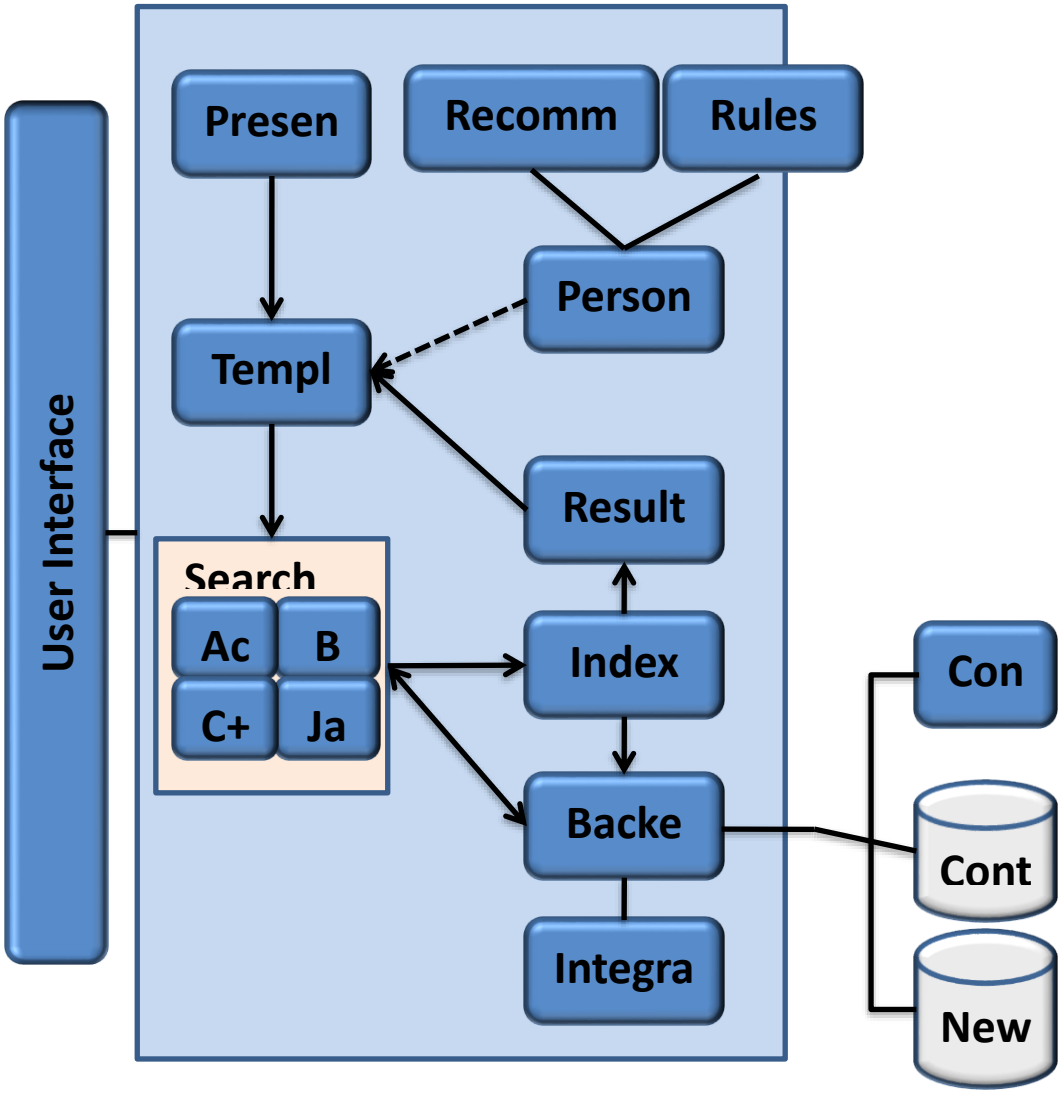


Figure SA-11: Universal Search Engine

Areas of consideration when selecting a search engine:

(a) Supported features

Some of the features to consider when selecting a search engine are:

(a) Results sorted by relevance

(b) Natural language queries (especially in Arabic)

(c) Search operators for greater control

(d) Secondary searches within the current results

(e) Search similar documents feature

(f) Metadata keywords search feature

(g) Metadata descriptions feature

(h) Search both structured and unstructured data feature.

(b) Support for customisation to map any meta-tag into any searchable attribute.

4.4.1.9 Integration Broker

There is no design consideration for this component.

4.4.1.10 Transaction Processing Monitors

There is no design consideration for this component.

4.4.2 Internet and Intranet Access

Internet and Intranet is commonly used to access information and e-services.

Areas of consideration when implementing Internet and Intranet Access:

(a) Support Different Client Devices

Access to information and eServices has to be designed to be delivered through browser in different client devices, in particular for the modern mobile devices.

(b) Use 'Push' Delivery Mechanism

'Push' or deliver information (e.g. news, alerts and reminders) to user without user requesting for it.

(c) Adopt A Consistent Presentation Design

User access via Internet website, Intranet or mobile applications have to be over a consistent interface that provides the clear identity of the government agency with standard navigation paths.

4.4.3 Telephony Access

While Internet access is pervasive, telephony access continues to be important as it gives an alternative access to information and e-services.

Areas of consideration for implementing Telephony Access:

(a) Enhance service access through IVR System, SMS and Fax

(b) Provide Call Centre Services.

4.4.4 Collaboration Management

4.4.4.1 Electronic Mail

There is no design consideration for this component.

4.4.4.2 Instant Messaging

There is no design consideration for this component.

4.4.4.3 Short Message Service

There is no design consideration for this component.

4.4.4.4 Collaboration Workspace

There is no design consideration for this component.

4.4.4.5 Video Conferencing

Please refer to TRM Platform Domain for details.

4.4.4.6 Enterprise Content Management (ECM)

According to Association for Information and Image Management (AIIM) International, the five components of ECM model are capture, manage, store, preserve and deliver. Under the manage category, the application areas are:

(a) Document management
(b) Collaboration
(c) Web content management, including web portals
(d) Record management
(e) Workflow/business process management.

As most of the application areas have been covered in this document, the focus will be on Document Management for this section.

Document management (DM) software provides a set of services for organizing electronic documents; managing content; enabling secure access to documents and unstructured data; routing documents and automating related tasks; and facilitating document distribution.

Areas of consideration when selecting DM software:

(a) Customisable Metadata

Metadata are the pieces of information that are captured together with a document into the Document Management System (DMS). As each agency may have different requirements for classifying their documents, it is important for the Document Management System to be flexible in allowing different Metadata to be captured. (Refer to Data Management Domain in IRM for best practices on Metadata).

(b) Access Control

Different DM products provide different ways of managing access control. The control can be by user and what he is permitted to do as well as by folders and documents. Agencies should examine the access control mechanism provided by the DM to assess if these fit their requirements.

(c) Versioning

Versioning is an essential feature of DM software. It provides a history of changes and allows incorrect change to be backed-out quickly. Different products offer different degree of control for versioning; some even allow versioning to be turned off, which is undesirable. In addition, the number of versions (e.g., 0-99) and sub-versions (e.g., 99.99) allowed varies from one product to another. This feature should be examined to ensure that the product meet your agency's requirements.

(d) Integration with email platform and office productivity software

Integration with the following:

(i) Filing of emails into DMS

(ii) Users to receive document tasks (such as reviews) via the same inbox as the email platform. A link to the document should be provided in the email rather than using the attachment feature

(iii) Filing of documents into the DMS through the office productivity software.

(e) Integration with Directory Service for user information

It is important to look for Document Management products that are capable of obtaining user information from a directory using LDAP protocol. If your organisation already has an enterprise directory, you should capitalise on this to reduce the additional efforts of creating another user information directory and having to maintain it.

(f) Compliance with legal regulations

If there is a requirement for documents kept in a Document Management to comply with any regulations (such as Royal Decrees and any government regulations), then there is a need to obtain certification from an authorised certification authority.

Lastly, AIIM has published a document entitled “Implementation Guidelines and Standards Associated with Web-based Document Management Technologies”. This document contains a number of guidelines, which may be useful to agencies implementing Document Management systems.

4.5 Technical Standards and General Standards

Please refer to Appendix SA-2 for the technical and general standards on Service Access Domain.

4.6 Best Practices

When implementing service access solutions, observe the following best practices:

(a) Standardise web browsers used in all agencies to minimise maintenance effort

(b) Constantly check and apply latest versions of patches to web browsers in agencies

(c) Ensure applications and e-services compatibility by avoiding the use of client-side technologies which are dependent on browser type and browser version.

4.6.1 Web Application Platform

4.6.1.1 Application Server

Best practices for application servers include:

(a) The administration service for the application server should not be started unless it is needed or it should be configured so that its access should be restricted to selected secure subnet.

(b) Enable services that are only required by the application server.

(c) Use clustering for load balancing and fail-over to increase reliability and performance.

(d) There may be performance issue if the application server is shielded from other components (e.g. Web Server or the database) by firewall. Where required, the firewall and other network devices should be tuned to compensate for the additional delay in transiting across components.

(e) As the number of threads affects the application server's performance, avoid the use of single thread model as this result in creation of an instance for each user and will overload the server very quickly.

(f) As data persistence technologies have become more mature, use Container-Managed Persistence (CMP) entity beans so that the persistence layer can take care of the database operations. This provides the benefit of developing entity beans that are more independent of the database. Use Bean-Managed-Persistence (BMP) entity beans only if CMP does not meet requirements. Similarly, use ADO.NET Entity Framework as it uses conceptual data model rather than physical relational data structures.

(g) Tune the connection pools for the application server, transaction parameters (like transaction serialisable), the server Operating System (OS) network parameters (like queue length, timeout, etc.) accordingly to the volume and characteristics of the application.

(h) For J2EE Application servers, tuning of the JVM is very important and will affect the performance of the application. Proper tuning of memory sizing, garbage collection settings etc. should be done with relation to application characteristics and design.

(i) The best practices for designing transactions in application server are:

i. Transactions should start from with the receipt of a user request and end with the return status of the request. It should not span user-thinking time at the screen/input device

ii. To optimise performance, the database transactions should be started and stopped at the application server level in the EJB container

iii. Transactions should be declarative and container managed to minimise the writing of codes to a transaction management service API. It will also reduce chances of errors and allow ease of changes to the transaction behaviour without substantial coding change

iv. If the business logic or rules are not sensitive, these can be located in the application server to improve performance and responsiveness

v. Do not manage transactions in client applications unless there are compelling reasons to do so. This is because client applications are subjected to interruptions or unexpected terminations and if you start and stop a transaction at the client level, you will risk consuming network resources while waiting for user actions, interruptions, resumption of client activity or timeout. You will also risk consuming process and network resources to roll back the transaction after timeout or termination of the transaction

vi. Watch out for potential deadlock for transactions involving database access/update

vii. If necessary, tune the database parameters like changing to row level locking, lock escalations values, frequency of deadlock scanning, lock timeout values, enable pessimistic locking, etc.

viii. Transaction monitor parameters like database connection pooling should be set according to application requirement.

4.6.1.2 Integration Server

The best practices for integration server include:

(a) Leverage on existing application server as integration server

Extend an existing application server into an integration server if there is an available application server that can be incrementally extended to meet the new requirement. The cost of extension and its available future growth potential must be evaluated against the cost of the new integration suite.

(b) Common schema for data interchange

A common schema for data interchange should be used when integrating systems across agencies. XML schemas (XSD) should be preferred over XML DTD (data type definitions) to define the data

interchange format because of its better extensibility, support for data types and namespaces. Please refer to IRM for data exchange standards and code tables.

(c) Use of XML Web Services

XML Web Services should be deployed as an alternative light-weight integration strategy instead of using it to substitute existing Enterprise Application Integration (EAI) solutions completely. This is because existing XML Web Services standards and products are still lacking in terms of transaction support and reliability, which are already quite established in traditional EAI solutions. Agencies should consider XML Web Services as one of their integration strategies with external agencies, or private sector organisations. This is less costly and reduces integration effort compared to traditional EAI implementations.

4.6.1.3 Database Server

There are a number of best practices described in IRM. In addition, the following are best practice for database servers:

(a) Independent Software Vendor (ISV) support

As more and more organisations buy off-the-shelf applications rather than develop in-house, the support of existing DBMS platform become a critical selection criterion for choosing the applications. Hence, organisation employing a 'buy' strategy for application should choose a DBMS that has wider ISV support.

(b) Cluster technologies for mission critical system

DBMS that support cluster technologies is recommended for mission critical systems that require high availability. Cluster technologies support failover capability: The automatic transfer of workload from one processor to another should there be an OS and hardware failure. In order to implement clustering for DBMS, the network and OS must be able to support it.

(c) Distributed Transaction Processing

DBMS that supports XA is recommended for system that requires distributed transaction processing (DTP). The XA is an X/Open specification for distributed transaction processing (DTP). It describes the interface between the global transaction manager and the resource manager (typically a DBMS). The XA Specification describes what a resource manager must do to support transactional access. Resource managers that follow this specification are said to be XA compliant.

4.6.1.4 Directory Server

The best practices for directory service include:

(a) Applications and operating systems should be directory-enabled

Applications that are directory-enabled can obtain an expanded set of user information from the directory service as appropriate. This will also help to modularise the user management functionality in applications.

(b) Make use of the directory service for authentication and authorisation

The directory is well suited to provide information on the level of security necessary. This facilitates user authentication and authorisation by making the resources available to the users, when the appropriate rights are in place.

(c) Synchronisation of multiple directories

In a typical environment where there are multiple directories deployed, the directories should be synchronised in order to improve data accuracy and integrity and eliminate duplicate, ghost and dormant accounts. Note that it may not be possible to synchronise the multiple user passwords in situations where the passwords are encrypted.

4.6.1.5 Transaction Processing Monitor

Transaction processing monitors are typically used in environments with a very high volume of transactions. A transaction monitor helps to manage these high

volume transactions and allows the systems to be more scalable. As a best practice, the direct use of TP Monitors is not encouraged. It should be used as part of an application server service. Figure SA-12 shows a recommended architecture for TP monitor as part of the application server service:-

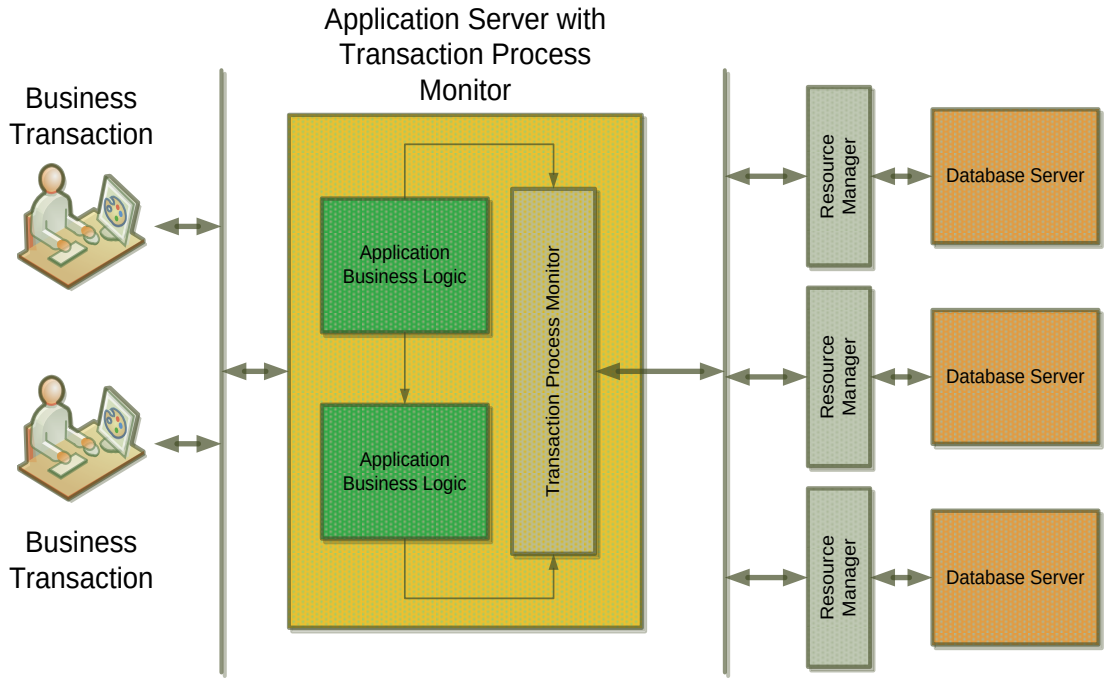


Figure SA-12: Transaction Process Monitor

4.6.2 Collaboration Management

4.6.2.1 Email

The best practices for email are:

(a) Single email infrastructure within an agency

Agencies should adopt a single email infrastructure to minimise integration and deployment complexity.

(b) Email servers, relays, gateways and filters should be dedicated

All unnecessary services should be removed from the server. Customised applications should not be executed on the server. Instead they should reside on a separate dedicated applications server.

(c) Rights to execute agents/scripts

As the poorly written agent/scripts could potentially bring down the mail server, execution of agents/scripts should be controlled and be available to only a limited group of users. The administrator should keep track of the list of users that are running agents/scripts on the mail server. The users should only execute thoroughly tested agents/scripts.

(d) Modems connected to the mail server

The mail server should not allow direct dial-in access. Remote users should instead dial into and be authenticated through the dedicated remote access servers (RAS).

(e) Inter-government mail routing

Inter-government mails should be routed within the Oman Government network. Only unclassified mails can be routed via Internet.

(f) Reduce mail database size

As the mail database file grows larger, the probability of mail database corruption increases. Hence, it is advisable to keep the mail database files small. This can be achieved through:

(i) Configure server to cap mail database size
(ii) Regular database compaction
(iii) Educate user on email archival.

(g) Limit Internet mail size and network bandwidth

The Internet bandwidth is usually shared for both email and web access. Dedicating a minimum and maximum Internet bandwidth for email traffic will help ensure that the emails continue to flow during web surfing peak hours. Large Internet emails consume Internet bandwidth and will likely bring down the mail filters and gateways. It is recommended to cap the mail size for incoming Internet mail. The size should be based on the agency's needs and the capacity of the mail filters and gateways.

(h) Stop open-relaying for Internet SMTP relays

To prevent spammers from misusing the Internet SMTP relays, sites should close off their SMTP relays to all but the predefined internal mail hosts.

(i) Filter Internet spam mails

Unsolicited commercial emails, or spam mails can be minimized by denying SMTP connection that originate from servers listed in publicly available DNS blacklists and reputation databases. These blacklists and reputation databases keep records of Internet SMTP hosts that are known sources of spam or those that permit third party open relaying.

(j) Perform virus scan and mail attachment filter

E-mails from Internet should be virus scanned and malicious codes should be removed. Email attachments that are popular means of carrying malicious codes (such as executable and macros) should be filtered. These detection tools should be constantly updated to detect new virus and file signatures.

(k) Keep up with security patches

Administrators should be kept updated on security issues for those related to the mail server, gateway, relays and their operating systems. They should subscribe to the vendor's security alert list or monitor their web site regularly for patches, test and apply them immediately.

(l) Monitor logs

System logs should be monitored regularly for any suspicious activity. Logs should be set as read-only, backed up regularly and should be archived for at least two years.

4.6.2.2 Instant Messaging (IM)

As there are still no unified standards for IM, agencies should seek alignment with the Messaging/Email Infrastructure for maximum interoperability with other agencies.

4.6.2.3 eParticipation

Introduce eParticipation through social media, online surveys, discussion forums and blogs.

4.6.2.4 Collaboration Workspace

The best practices for collaboration workspace are:

(a) Leverage on existing messaging infrastructure

Agencies should use existing solutions available for their messaging infrastructure (refer to Email, Instant Messaging and SMS) to implement collaboration workspaces. This will reduce the complexity in integration between the workspace solution and messaging.

(b) Dedicated server

Similar to email server, all unnecessary services should be removed from the server. The mail service should not be executed on the same server as the collaboration workspace server.

(c) Perform virus scan on documents

The server should perform virus scan on all documents prior to saving them to the server.

4.6.2.5 Video Conferencing

Best practices for video conferencing are:

(a) Interoperability testing

If products from different vendors are to be used together, it is recommended that a test be conducted to ensure complete interoperability, even if the vendor state compliance with the standards listed above.

(b) Bandwidth provisioning

Typical video conferencing session consumes about 256Kbps to 1 Mbps of bandwidth. Provide for such bandwidth prior to the video conferencing

session. Make sure also that the corporate firewall setting allows such a connection to take place.

(c) Audio setup

Voice is actually more important than video in a videoconference session. Plan for full duplex (bi-directional) audio and the best microphones and speakers that can fit into the budget. Do also sure that echo and noise cancellation is a feature in the audio system.

4.7 Obsolete Technologies

Government agencies have to ensure that they do not have any obsolete technologies in this domain. Please ensure compliance by referring to OeGAF Obsolete Technologies Compliance List.

5. Service Integration Domain

5.1 Intent

The Service Integration Domain defines the various service integration technology categories, technology components and associated standards. It highlights the key architecture design considerations and recommends best practices for implementing service integration solutions.

5.2 Domain Design Principles

When designing a service integration solution, the following design principles should be observed:

Principle 1: Enable the Inter-operability of Multiple Technologies

Integration provides a bridge between the heterogeneous operational applications and platforms. An effective architecture ties together the mix of platforms, operating systems, transports, and applications.

Principle 2: Leveraging on Existing Technologies and Investment While Integrating Different Application Systems

Service Integration has to take into account the need to use existing workstations, peripherals and existing transports to access existing and new applications. The idea is to leverage on existing investments to connect the various operational application systems and data.

Principle 3: Use Existing Integration Solutions Whenever Possible

Instead of building a new integration technique from scratch, use an existing solution that answers the specific integration needs of an application system.

Principle 4: Reuse Common Services to Ensure Efficient Integration and Reduce Development Time

Provide access to common services (such as accessing frequently-used data and access control to data / applications) that can be reused and shared, thus

reducing development costs. Reduce the resources spent on developing and maintaining "islands of applications" which include redundant code. Application developers can focus on new work rather than re-work.

Principle 5: Minimise the Impact to Existing Application Systems

As much as possible, the integration solution should enable new applications to use existing resources with minimal disruption to existing applications. Where possible, use non-invasive techniques for integration.

Principle 6: Use Standard, Matured and Widely Adopted Technologies Across Applications Whenever Possible

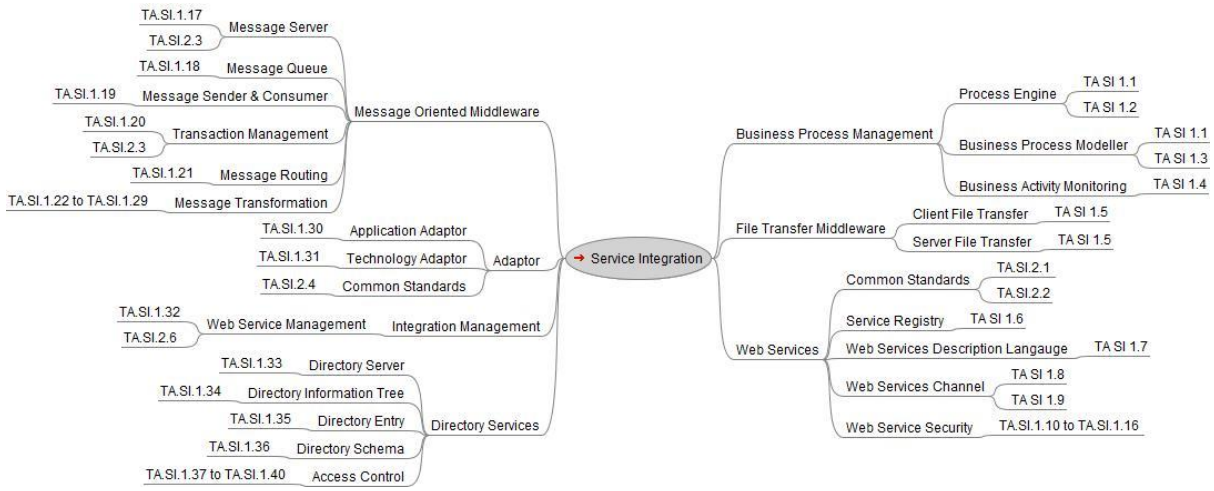
Use standard, matured and widely adopted integration technologies. Limit the heterogeneity of the technology used in order to simplify integration and enable migration to future technologies.

Principle 7: Provide Maximum Flexibility to Integrate Heterogeneous Systems

Implement standards-based service integration technologies that provide maximum flexibility for both existing and future needs.

5.3 Technology Categories and Components

Figure SA-15 shows the association between the technology categories, technology components and its relevant standards.



(need to replace diagram)

Figure SA-13: Mapping of Categories, Components and Standards for Service Integration Domain

Table SA-12 describes both the technology categories and technology components of the Service Integration Domain.

Technology Categories	Technology Components
Business Process Management (BPM) A Business Process (BP) is a collection of related, structured activities that provides a service that meets the needs of a client. BPM is a technology category which comprises enabling components for automation of BP.	**Process Engine** The Process Engine is the run-time platform for executing process-based applications including business rules. Commonly used runtime languages are Business Process Execution Language (BPEL) and XML Process Definition Language (XPDL).
	Business Process Modeller This is the design time platform for designing and simulating Business Processes. The commonly used design-time notation is Business Process Modelling Notation (BPMN).
	Business Activity Monitoring (BAM) This is the business analytics tools which are designed specifically to analyse business processes, enabling managers to identify business issues, trends and opportunities with reports and dashboards and react accordingly.

Technology Categories	Technology Components
File Transfer Middleware This refers to the components used to transfer data files over the network. File transfer can be carried out between two servers, and also between a client and a server.	**Client File Transfer** This component initiates the connection, requesting to either receive or send required data.
	Server File Transfer This component contains either the data file to be deposited or the data file to be extracted. It also manages the access rights to the data repository.
Web Services Web Services are defined as self-contained, self-describing, loosely coupled software components that can be published, discovered, and invoked over a network.	**Service Registry** Like a directory, the Service Registry lists all the available Web Services that can be consumed. It allows a Web Service to be published, searched and invoked.
	Web Service Description Language (WSDL) The WSDL is a language that allows the Web Service Provider to describe the functions of the service. It also allows the Web Service Consumer to understand the service, and how and when to invoke it.
	Web Service Channel Web Service channel is the communication method between the Web Service Provider and Web Service Consumer. Being platform independent, the messages are constructed using XML documents which contain metadata and its corresponding contents. The runtime binding establishes a dynamic relationship between the Web Service Consumer and Web Service Provider, creating a self-

Technology Categories	Technology Components
	contained service that maintains its own state (be it stateful or stateless).
	Web Service Security Web Service Security is a set of mechanisms to implement secured Web Services. It describes the various ways to ensure the confidentiality and integrity of Web Services.
Message-Oriented Middleware (MOM) This is an infrastructure that transfers messages over the network. Data is enveloped into messages so that they are platform independent.	**Message Server** Message Server is a middleware that handles messages sent for use by other applications / programs, using a messaging Application Program Interface (API). A Message Server typically houses the message queues, and manages the transactions between a message sender, message queues, and the message consumers.
	Message Queue Message Queue provides the data transmission method between application systems. Messages are deposited and received through Message Queues. This technology provides transparent, interoperable and robust message exchange.
	Message Sender and Consumer

Technology Categories	Technology Components
	Message Sender is a program that sends information to another program via the Message Server. A Message Sender program uses the messaging API, creates a connection to the Message Server and writes the message to the Message Queue. A Message Consumer program waits for messages in a Message Queue. When a message arrives, the consumer program processes the message and commits the transaction.
	Transaction Management This technology allows efficient processing of transactions across various heterogeneous systems. Messages are coordinated and sent to respective distributed systems for processing. The Transaction Manager ensures data integrity and the completion of transactions.
	Message Routing It provides location transparency for the message processing. Based on the message header (including message priority), messages can be routed to improve the efficiency processing of data and transactions. Message Routing is based on pre-defined rules and

Technology Categories	Technology Components
	algorithms. It is useful for subscription-based and high priority messaging.
	Message Transformation This technology provides the ability to transform from one protocol to another which is important for interoperability. The MOM can receive a message in one protocol and transform into another before sending it out.
Adaptor This software bridges the differences among various application software technologies and protocols. It is a simple solution that usually 'eliminates' the gaps between legacy and new application technologies.	**Application Adaptor** This software allows connectivity among customised product-based application products. Customised Off-the-Shelf (COTS) applications may require specific application adaptors so that the data and application functions can be integrated with other applications.

Technology Categories	Technology Components
	Technology Adaptor This software typically bridges the differences between legacy applications and new technologies. Mainframe application adaptor to access a Web Service is an example.
Integration Management Integration Management software provides monitoring and governance capabilities for integration solutions.	**Web Service Management (WSM)** In particular for Web Services, a WSM would provide the following functionalities: (a) Lifecycle Management – for the provisioning, versioning and terminating of the Web Service (b) Quality of Service (QoS) – monitors and measures the availability and performance of published Web Services (c) Operations Management – monitors, alerts and provides audit logs on the Web Services execution. Only with proper operations management can Web Services be implemented and maintained effectively.
Application Integration	**Application Interface** This involves the invocation of specific business functions of a specific API or pre-determined protocols. This allows the reuse of the functionality. The use of distributed objects or components is also included.
	Workflow

Technology Categories	Technology Components
	Workflow software is a vehicle for automating business processes and tracking their status. It enables work to be assigned, routed, approved, acted upon, and managed electronically through system-controlled rules.
	Data Integration Data integration is the process of sharing or merging data from two or more distinct software applications. It includes the following means: (a) Data sharing via direct database connection or database replication (b) Data transfers via batch file transfers
	Message-Oriented Integration Messaging is an event-driven communications layer that allows applications to transfer information securely and reliably. It enables reliable communication between heterogeneous applications and supports asynchronous communication. Reliable transmission of message is assured.
	Service-Oriented Integration Service-oriented integration rides on top of the traditional application interfaces and message-oriented integration to hide the

Technology Categories	Technology Components
	complexity and abstract out the interfaces in order to enable a more flexible, platform independent and robust integration environment.
	Process-Oriented Integration The integration approaches used are becoming more aware of its role in the large scheme of a business activity or business process. Process-oriented integration is in recognition of the convergence of business process with traditional integration models.
Directory Services A directory is an information source used to store information about objects, such as users, applications, and network resources, in a hierarchical tree format that can be set up to represent an organisation chart. A directory service refers to both the information source (directory) and the services making information available to the users.	**Directory Server** The actual information store that allows clients to query the directory.
	Directory Information Tree (Namespace) Organisational structure for all the entries.
	Directory Entry A directory entry is a collection of attributes with a name, called a Distinguished Name (DN). The DN refers to the entry unambiguously.
	Directory Schema Information about how data is organised – metadata such as object classes, attributes, matching rules – and stored in the directory schema.

Technology Categories	Technology Components
	Access Protocol The protocol for clients to access directory servers.

Table SA-12: Service Integration Technology Categories and Components

5.4 Architecture Design Considerations

5.4.1 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is an architectural style, design style and design principle for application development and integration. It is intended to achieve loose coupling among interacting applications or services.

These services interoperate based on a formal definition (or contract) which is independent from the underlying platform and programming language. The application software components become very reusable because they are standards-compliant and are independent from the underlying implementation of the service logic.

Areas of consideration when designing an SOA based solution:

(a) Leveraging on current technology investment

The solution should provide opportunity to consolidate similar application functionalities leveraging on current technology investment as far as possible (e.g. through the use of Web Service, adaptors and / or screen scraping)

(b) Flexibility

The solution should structure ICT applications based on services in such a way as to facilitate the rapid reconfiguration of business processes

(c) Agility

The solution should allow users to respond to changing business requirements through quick deployment of new/enhanced applications.

Figure SA-14 below illustrates the reference architecture of a typical SOA implementation today.

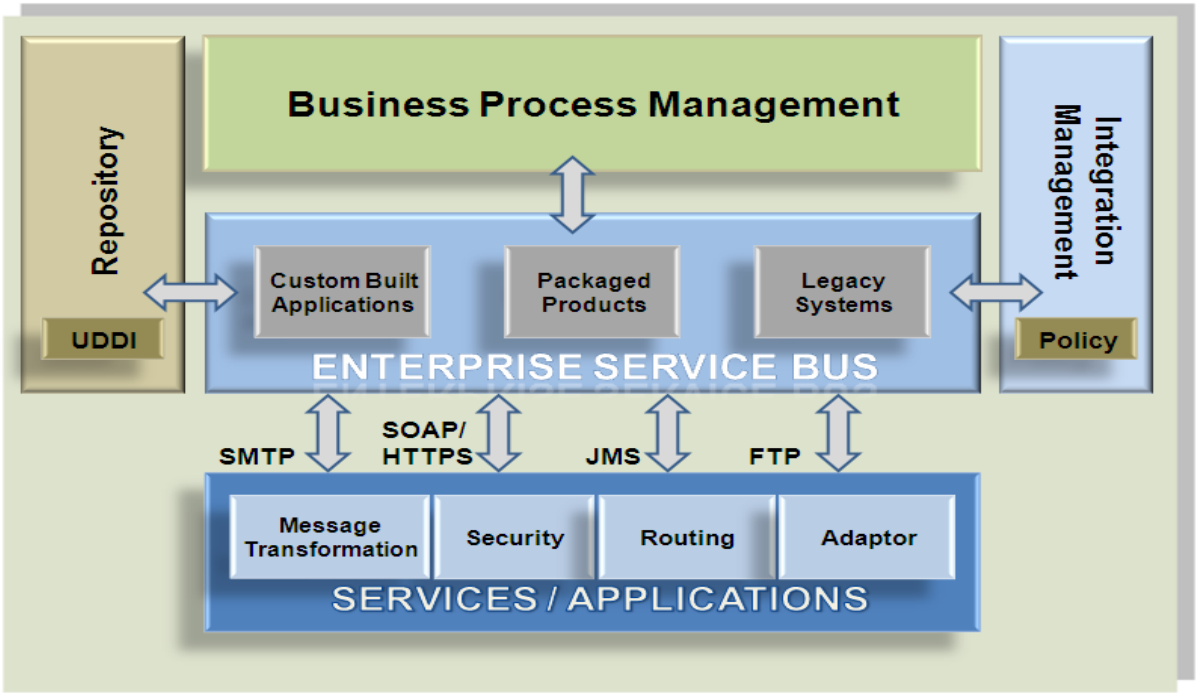


Figure SA-14: Overview of SOA Reference Architecture

5.4.2 Business Process Management (BPM)

BPM is a technology category which comprises enabling components for automation of business process.

Areas of consideration when implementing BPM:

(a) Segregate deployment environments

Typically, there are four environments required during the course of BPM implementation (i.e. Development, Test / QA, Staging and Production)

(i) Development: This is primarily used for developing BPM solutions. All unit tests, bug fixes are done in Development

(ii) Test/QA: The environment is used primarily for deployment of solutions for testing of features and overall functionality and user-acceptance of solutions

(iii) Staging: This environment is a duplicate of the Production environment, where all fixes are tested in an environment that is identical to Production, before migration into Production

(iv) Production: This is the live environment where actual business transactions take place.

(b) Decide on an architecture option that meets business requirement

There are four architectural designs for BPM implementation, namely:

(i) Single-tiered or Standalone: This option provides a single and simple deployment for all the BPM components (BPM Engine, Application Server, and Database Repository) on a single machine. In addition, this option provides a simple administration, less overhead, and limited transactional capabilities. This option is for simple to moderate BPM deployments

(ii) 2-tiered: This option provides a two-tiered architecture deployment where on the one hand, the BPM engine and the Application Server are installed on one server and on the other hand the database repository is installed on another server. This usually happens when the ICT infrastructure already has some database instances that can be leveraged. This is a mid-level architecture option where there is slightly more overhead and more transactional capabilities. This option is for moderate BPM deployments

(iii) 3-tiered: This option provided dedicated services for all BPM components. In addition, this option provides more complex deployment and administration; there is much more overhead but it is scalable

(iv) Multi-tiered: This is the most complex architecture where BPM components are divided into multiple dedicated servers for large-scale BPM deployments in an established ICT infrastructure that includes clustering, load-balancing, and/or fail-over.

The selection of these options for each environment depends on different factors such as existing ICT infrastructure, budget, and solutions to be deployed. The most commonly implemented architecture option is the 3-tiered architecture.

5.4.3 Enterprise Service Bus (ESB)

An ESB is a distributed service reference architecture, built based on open standards, which delivers messaging middleware, intelligent routing and XML transformation, in conjunction with a flexible security framework and a management infrastructure for deploying and monitoring the services. An ESB allows services - mainly data and application services - to be searched and dynamically consumed. More importantly, an ESB provides a loosely coupling implementation mechanism between applications, between data, and between data and applications. By allowing transformation of messages in different formats, a change in the application or data format would not require explicit change in the applications that require the data or logic from the application/data affected.

Areas of consideration when implementing ESB:

(a) Use ESB only for complex, enterprise integration requirements

(b) Implement both synchronous and asynchronous transport protocols and service mapping (locating and binding) for flexibility

(c) Use different message routing features such as static / deterministic routing, content-based routing, rules-based routing and policy based routing

(d) Use logical partitioning and proper name space for message queues

(e) Use adaptors for integration with legacy or proprietary-based applications. Adaptors can be broadly classified into technology adaptors and application adaptors:

  (i) Technology adaptors provide connectivity via standard transport protocols (e.g. HTTP, FTP and SMTP)

  (ii) Application adaptors provide connectivity to packaged software by invoking their APIs directly

(f) Implement a security module to authorise, authenticate, and audit the use of ESB which enforces non-repudiation and confidentiality. This security module would be independent from the other applications

(g) Enforce assurance in delivery of messages.

5.4.4 Repository

A repository is the component in an SOA architecture used to manage design-time and run-time artefacts and assets generated to support each service. This includes functional specifications, message meta data, and service contracts as shown in Figure SA-14.

Areas of consideration when implementing repository:

(a) Start with a simple repository when implementing any type of integration. It is more important for the repository to provide the ability to track all integration jobs, file transfers, Web Services or messages. The repository can be in the form of an Excel spreadsheet

(b) Over time, as the volume of integration increases, consider implementing a UDDI which is a set of specifications that describes how a registry stores information about Web Services and how the registry can be accessed

(c) Consider implementing Lightweight Directory Access Protocol (LDAP) as an application protocol to query and modify directory data running over TCPIP.

5.4.5 Integration Management

This component provides governance capability in an SOA. SOA governance is a set of activities related to exercising control over services in an SOA. These include managing of Quality of Service (QoS) policies of all service interfaces within the SOA.

Areas of consideration when implementing Integration Management:

(a) Regardless of the integration methods (e.g. files transfer, Web Service, message or adaptor), it is necessary to manage the integration execution. As integration involves two parties, there must be a documented agreement on the integration services.

Consider Service Level Agreement (SLA) for the core and important integrations. Key information to be captured in the SLA includes Service Owner, Service Availability and Service Access Rights.

(b) Use monitoring tools to check on basic message queue objects like queue depth, connections and error queue, etc.

(c) For Web Services, implement Web Service Management (WSM) to monitor and track Web Service performance.

5.4.6 Application Integration

In order to achieve a level of cohesiveness and interoperability, there is an increasing need for applications to be integrated with other applications to share data and processing. User requirements are inherently too complex and dynamic for any single design team to provide the entire solution. Hence, it is

necessary to have integration strategies to achieve the integration of potentially homogeneous and also multiple heterogeneous solutions.

An integration strategy is required for the development of large, complex application. An integration strategy is also required when there a number of interfaces to data and business processing (either within the government agency or with other government agencies). An integration strategy may use one or more interface tiers as shown in Table SA-13.

Interface Tiers	Examples of Integration Methods
Data Integration	File transfer, data replication, etc.
Application Interface	Common Object Model (COM), Common Object Request Broker Architecture (CORBA), XML Web Services, etc.
Message-Oriented Integration	Message queues
Service-Oriented Integration	Web Services
Process-Oriented Integration	Business Process Management(BPM)

Table SA-13: Types of Interface Tiers

5.4.6.1 Data Integration

Data integration involves combining data residing in two or more distinct software applications to create a more highly functional enterprise application. Reference can be made to Information Reference Model (IRM) for data integration details. This is typically achieved via the following means:

(a) Data sharing via direct database connection or database replication

(b) Data transfers via batch file transfers

Areas of consideration when selecting data integration methods:

(a) Network where the application resides

If the application is residing on another network, direct database connection will typically not be possible. In such cases, consider using file transfer middleware

(b) Data integrity and consistency

If data integrity and consistency is crucial, use less intrusive methods (no direct data access to application) such as file transfer middleware

(c) Data Timeliness

If real time update is required, consider data access methods or protocol such as Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC)

(d) Database product of the application

Database replication is usually product-specific. Hence, it is preferred that both applications use the same database product if one wants to use data replication.

A sample evaluation of the options and the different data integration methods are shown in Table SA-14. Related information is also provided in Information Reference Model (IRM).

Data Integration Method	Security	Data Integrity and Consistency	Data Timeliness
File Transfer Middleware	Does not require direct access to database by other applications (i.e. less intrusive).	Each application will need to handle the integrity of the file upload to the database. Application only needs to ensure the integrity of the application data is not compromised during	Applicable for applications that do not require real time update.

Data Integration Method	Security	Data Integrity and Consistency	Data Timeliness
		upload from the flat file.	
Data Access Methods & Protocols	Opens up new opportunities for data corruption and security breaches. Direct access to database increases vulnerability to potential mis-use.	Applications that require direct access to the database must ensure all appropriate integrity checks are in place at the application level.	For applications that require real time update.
Database Replication	Can make use of secured replication. However, it is not firewall friendly.	Synchronisation of replication activities is important to ensure that it meets business objectives.	Provides real time or batch updates.

Table SA-14: Data Integration Strategy

5.4.6.2 Workflow

There are three major types of workflow products:

(a) Embedded Workflow

Embedded workflow comes as part of a business application package (such as ERP or CRM) and is embedded within the business application – this embedded workflow generally only addresses the needs of the particular business application and does not extend its reach further. Besides business application packages, embedded workflow are provided as part of other software like Document Management software and Web Content Management software.

(b) Independent Workflow

Independent workflow software comes from third-party workflow vendors that supply standalone software packages designed to complement and coexists with an organisation's existing IT infrastructure. The software generally comes equipped with good design tools, and may offer add-on EAI-type functionality. It typically involves repetitive processes (processes that obey a standard set of rules and policies that are applied to every work item). The product is built to be endlessly definable, adjustable and fluid. The design methodology is top-down, driven by a mapped business process and drilled down to specific sets of rules.

(c) Infrastructure Based Workflow

Such software provides a layer of functionality on top of an infrastructure system, such as messaging infrastructure and application server.

Areas of consideration when selecting workflow software:

(a) Routing mechanisms

The routing mechanisms can be sequential, parallel or rule-based. As not all workflow software is capable of supporting all of these routing mechanisms, it is important to select one that will meet application-specific requirements

(b) Task notification

It is important to ensure that the workflow software can integrate with your email platform so that the task notifications can be received via the same inbox as the email platform

(c) Integration with directory service for user information

The workflow should be able to integrate with directory service for user information

(d) Workflow definition tools

It is crucial that the workflow software should provide tools to ease the design and maintenance of the workflow instructions

(e) Development support

If customisation is required, agency should evaluate the toolkits provided by the vendors

(f) Integration to BPMS

Business Process Management (BPM) is a change management and system implementation methodology to aid the continuous comprehension and management of business processes that interact with people and systems both within and across organisations. A BPM System (BPMS) is the technology solution component of BPM approach to change management and system implementation and workflow software is one of the essential components of a BPMS. It is therefore essential to consider the integration to BPMS if the overall need is for BPM.

5.4.6.3 Application Interface

In this method, the sharing of data is achieved by invocation of specific business functions via specific Application Programming Interface (API) or pre-determined protocols. The APIs have to be developed by the respective applications and made available to the rest of the applications. Such APIs may include the use of distributed objects or components.

Areas of consideration when implementing application interface:

(a) Platform of the application

If the applications are developed in Java, consider using JNI (Java Native Interface). It is simple to use and is faster than other approaches

(b) Portability of API

If the API needs to be portable and language neutral, consider using CORBA (Common Objects Request Broker Architecture).

5.4.6.4 Message-Oriented Integration

Messages are an excellent form of integrating or exchanging event-driven data. There is no design consideration for this component.

5.4.6.5 Service-Oriented Integration

Please refer to TRM Service Integration Domain, Section 5.6.1 for design considerations.

5.4.6.6 Process-Oriented Integration

Please refer to Section 4.5.2 above for design considerations.

5.5 Technical Standards and General Standards

Please refer to Appendix SA-3 for the technical and general standards of Service Integration Domain.

5.6 Best Practices

5.6.1 File Transfer Middleware

(a) Large number of small or medium sized files should be packed into a single file for better performance, especially over slow network

(b) Use ASCII mode of transfer for text files

(c) Use Binary mode of transfer for non-text file (e.g. executable files)

(d) Open the FTP ports at the firewall only upon request for connecting to external FTP servers

(e) Ensure only authorised users have access to the right FTP directory

(f) FTP servers should be in DMZ

(g) Maintain regular housekeeping of files on the FTP servers.

5.6.2 MOM

(a) Use MOM only when integrating many application systems especially when involving legacy system

(b) When selecting a MOM solution, focus on performance, ease-of-use, and types of protocol supported for network and security.

(c) Avoid using bulk delivery mode for delivery of messages. The large amount of data in bulk delivery can potentially clog up the MOM infrastructure and degrade performance.

5.6.3 Integration Management

(a) Explore the feasibility of extending existing application servers into an integration server

(b) Use a common schema to enable effective data exchange

(c) Design Web Services to an appropriate granularity. Avoid creating too fine-grained Web Services because of the network overhead incurred for each web service invocation

(d) Assess the readiness of participating parties before implementing web service.

5.6.4 Application Integration

Best practices for application integration include:

(a) Security & performance issues with data integration

Data integration is probably the least secure of all the integration solutions. For example, for direct data access, the data sources must be opened to other applications and often to the outside world. Performance may also be an issue if large movement of data is required during synchronisation or file transfers.

Some best practices are:

(i) Keep sensitive data out of data integration solutions

(ii) Split databases according to the sensitivity of the data

(iii) Use DBMS access control features (judicious use of access controls by user and password can limit the vulnerability of the data. Create new user accounts and passwords for the data integration programs, restricting read and write access to those tables that need to be read or written by the integration programs)

(iv) Avoid public networks - use Oman Government Network or VPN technology

(v) Manage data access and usage

(vi) Automated data transfer may have capacity issues if it is not monitored. Data replication may work well in the beginning but transfer times may increase with increase in data volume. Monitor the replication and think of alternative if performance becomes a concern

(vii) Limit transformations. Transformations of data require processing time and will have impact to the performance.

(b) Reduce the number of point-to-point integration

The number of connections to maintain point-to-point integration increases exponentially when the number of integration points increases. Agencies should avoid point-to-point integration whenever

possible. Instead of having multiple applications establishing connections among themselves to exchange data, a data hub can be built to act as a gateway between these applications. These applications can then only need to integrate with the hub and not with each individual application. This eliminates the need for point-to-point integration among applications.

However, for multiple applications to access a single hub would mean that the hub must be able to take the load. Performance and scalability requirements must be taken into serious considerations.

(c) Design application for “integration agility”

The state of the application determines the degree of integration effort required. Applications must be designed upfront to allow “easy” integration. For example, each tier (business, data, data access, transport tier etc.) must be clearly segregated. These various “components” are then integrated together. Such loose coupling allows for easier integration. Tightly coupled applications suffer from difficulty in integration, as it normally requires application changes.

5.7 Obsolete Technologies

Government agencies have to ensure that they do not have any obsolete technologies in this domain. Please ensure compliance by referring to OeGAF Obsolete Technologies Compliance List.

APPENDIX SA-4 – Object Oriented Programming

The earliest OO methodologies – sometimes known as first generation OOAD methods – tended to adapt structured techniques such as functional decomposition and data flows to creating objects, while "second generation" methodologies approach analysis and design from the perspective of objects, transactions and messages. The third generation method, however, attempted to integrate a number of other methods. One example is the Unified Method that is an amalgam of the Booch (Grady Booch's methodology), OMT (Object Modelling technique from Rumbaugh) and OOSE (object-oriented software engineering from Jacobsen). Table SA-15 describes the traditional method (structured method) and the Unified Method:

OO Methodology	Description
Structured Method www.ieeexplore.ieee.org	The structured method divides an application development project into modules, stages and tasks. It sets out a waterfall view of systems development, in which there are a series of steps, each of which leads to the next step. Some of the notations more commonly used are Data Flow Diagrams (DFD) and Entity-Relationship Diagrams (ERD). A DFD is a graphical technique depicting information flow and the transforms that are applied as data move from input to output. It is incorporated in the analysis and design methods by Yourdon, DeMarco, and Gane and Sarson. An ERD is a data modelling technique that creates graphical entities, and the relationships between entities, within an information system. An entity can be a person, object, place or event for which data is collected. The de facto standard is Peter Chen's ERD.

OO Methodology	Description
Unified Method Cetus-links www.cetus-links.org/oo_ooa_ood_methods.html OMG www.omg.org UML 2.0 www.agilemodeling.com/essays/umlDiagrams.htm	The Unified method encompasses the OO methodologies from Booch, Rumbaugh and Jacobsen. This is deemed to be the method widely adopted by the industry. The notation is Unified Modelling Language (UML), a graphical mechanism for specifying, visualising, constructing and documenting the artefacts of a software system. The Object Management Group (OMG) endorses the UML. UML 2.0 supports the Model-Driven Architecture (MDA). In MDA, the models become "executable", with codes being generated automatically. The "logic" in the models can be verified before the codes are generated. With MDA, much of the development effort may be shifted to the level of the model.

Table SA-15: Development Methodologies

There has been a gradual shift towards process orientation in the software development paradigm to better align business goals with ICT systems. Understanding today's complex business processes is a significant challenge. Awareness of the details of the business process flow improves the quality of the requirements assessment for system development. Such an awareness can be achieved through business process modelling which is an activity performed by business analysts within a company. Analysts use modelling tools to depict both the current state of an organisation and the desired future state. The activity of modelling a business process usually predicates a need to change processes or identify issues to be corrected. With advances in technology, the vision of these models becoming fully executable and capable

of round-trip engineering, is becoming closer to reality every day. Supporting technologies include Unified Modelling Language (UML), Business Process Modelling Notation (BPMN) Model-Driven Architecture (MDA), and Service-Oriented Architecture (SOA).