452

EMS for the 21st Century

System Requirements

Working Group
D2.24

February 2011

EMS for the 21st Century System Requirements

Working Group
D2.24

Members

Richard Kalisch (US) – (Convener), Rolf Apel (DE), Jay Britton (US), Joe Bucciero (US)
Horia Stefean Cimpeanu (RO), Scott Coe (US), Kendall Demaree (US), Ebrahim Vaahedi (CA)
Gary Finco (US), Eric Fleuret (US), John Gillerman (US), Doug Goodwin (NZ), Dan Hazelwood (US)
Scott Heard (NZ), John Holt (US), Philip Johnson (UK), Walter Johnson (US), Mladen Kezunovic (US)
Anders Larsen (NO), Thierry Lefebvre (FR), Lucas Malesku (CA), Sergio Morand (BR), Jay Moser (US)
Khosrow Moslehi (US), Scott Neumann (US), Lars-Ola Osterlund (SE), Joerg Mueller (DE)
Lars Nordstrom (SE), Tony Patterson (US), Rodolfo Pellizzoni (AR), Eric Richer (CA)
Hiram dos Santos (BR), Augusto Sellhorn (US), Sharon Xia (US), Paul Skare (US)
Alain Steven (US), Jean-Pierre Turgeon (CA), David Ulmer (US), Xiaofeng Wang (US)

ISBN: 978-2-85873-141-1

Table of Contents

1. Introduction

1.1 Background

Energy Management Systems (EMS) and real-time grid operations systems, in general, are highly specialized, real-time, computer based control systems that were first introduced in the late sixties to assist system operators in managing the power grid reliably and efficiently.

Historically, achieving real-time computer performance at a reasonable cost remained a key limiting factor as industry requirements consistently exceeded the capabilities of commercially available hardware and software platforms. To overcome real-time performance limitations, system vendors devised proprietary software solutions that were expensive to develop and maintain. In addition, a lack of industry coordination and convergence in technical requirements led to highly customized software solutions thus preventing cost-effective productization of software applications and impeding migration of data models and applications to newer EMS System platforms. This resulted in high Total Costs of Ownership (TCO), and costly replacements of EMS Systems. Frequent schedule delays, costs overruns and high support costs created an unfavorable business model that forced a number of system vendors to exit the EMS System market. Through the nineties, advances in the computer industry enabled some of the surviving system vendors to increase the standardization of their product offering lowering their costs with substantial costs savings to their customers.

The emergence of deregulated energy markets lacking a standard market design led to a new wave of custom business applications interlaced with traditional EMS System applications. The new Market Management System (MMS) application projects typically involved regulatory uncertainty, changing requirements, and tight schedules contributing to inefficient and costly deployments.

In addition, internet technologies enhanced systems connectivity and interoperability substantially increasing the vulnerability of mission-critical systems on legacy platforms designed prior to the internet revolution and cyber attacks.

Consequently, the current state of the real-time grid operations systems industry can be summarized as follows:

- System operators rely on a limited number of specialized system vendors for the supply and maintenance of their real-time systems.

- The current Systems business model is marginally viable because of the limited market size and the lack of standardized requirements. This results in high development and support costs per installation that are incurred both by vendors and end users.

- Systems software platforms are monolithic and proprietary restricting opportunities for system expansion particularly with integration of third party products.

- The lack of critical mass and high costs associated with supporting the application platform reduces vendors' abilities to invest substantially in new functionality and applications.

- The prevailing pre-internet architectural design makes existing systems platforms highly vulnerable to cyber attacks and prohibitively expensive to upgrade.

- Most data models representing the electricity network are not standardized, are inconsistent from application to application, and vary from vendor to vendor. This results into high development and maintenance costs, the inability to easily mix and match applications from different vendors, and expensive migration of the data model and displays to newer platforms.

- The multiplicity of graphical user interfaces within a multi-vendor solution results in substantial user training and system operations complexity.

1.2 Objectives and Scope

The objective of the CIGRE D2.24 Working Group is to develop a common, world-wide vision for the next generation of Energy Management Systems (EMS) and related real-time grid and market systems. The scope of this set of requirements includes the real-time and near real-time systems which are used in support of the operation of the power grid and associated energy markets. Throughout the remainder of this document we will refer to these systems as the EMS/MMS Systems. This document outlines the broad business, functional, and technical requirements that the next generation of EMS/MMS Systems should meet.

This document is intended to provide EMS/MMS System software developers and system integrators with a common, world-wide specification for the future architecture to drive future product development. It is recognized that substantial time and effort will be required for system suppliers to migrate their product lines to this new architecture. By providing suppliers with a long-term vision, it is expected that the development process can be managed so that the costs to achieve this vision will be minimized through standardization and coordination of the world-wide procurement process.

To achieve the above objective, this document provides a set of suggested requirements and design principles for future EMS/MMS architectures leveraging existing IEC standards in support of improved system interoperability and modularity. It describes an end-state that is intended to become a standard component of future Requests for Proposals (RFPs). This will provide the industry (end users, system integrators, software developers and consultants) a consistent end-state to migrate towards in a consistent, cost-effective and coordinated fashion. It is also envisioned that, through standardization of the low added value architecture components, greater component reusability will be achieved enabling vendors to focus their efforts on the development of new and innovative functionality.

To ensure the technical viability and cost effectiveness of the future architecture requirements, these requirements have been developed by utilities in coordination with the major international system suppliers and consultants. They have provided valuable feedback and recommendations in an unprecedented spirit of cooperation.

It is not the intent of this document to be so prescriptive that it would stifle a system vendor's innovation and need for differentiation. On the other hand, the specified architectural design need to be sufficiently prescriptive to ensure the achievement of one of the major goals of this effort: a high level of interoperability between vendor platforms and applications, both at the inter-control center level, and within multi-vendor control center solutions. We have tried to achieve such a delicate balance.

1.3 Systems Architectural Vision

It is important to understand that the architecture requirements presented in this document are based on the following salient characteristics which are core to meeting the objectives of this working group's efforts.

- **SOA Adoption**. A component-based Service-Oriented Architecture (SOA) is a core concept of the architecture which is built around highly-modularized and reusable components. Standardized interfaces will facilitate easier integration of third-party software components. The de-coupled nature of the architecture will facilitate replacement or addition of new components to minimize upgrade and replacement effort and cost.

- **CIM Compatibility**. System vendors and third-party application providers will adopt the Common Information Model (CIM) developed by EPRI and industry participants. This will enable system operators to "mix-and-match" best of class applications, and reduce dependence on a single system vendor.

- **Built-in Security**. The architectural will be built around security requirements from the ground up making the system more secure and resilient to cyber attacks.

- **Platform independence**. The architecture will be abstracted from the physical hardware infrastructure and underlying software platforms which will facilitate the implementation of multi-vendor solutions and the migration of the business logic to future technologies.

- **Unified Graphical Interface**. The architecture will provide operators with an application-independent user interface to facilitate the implementation of workflow based displays integrating data and control from multiple applications.

1.4 Future Expansion of Architecture

This architectural requirements document is intended to be an evolving document. Starting as a top-down, high level document, it will evolve into a specification sufficiently detailed and prescriptive to ensure that the key business objectives and interoperability requirements are achieved. It is expected that the document will expand at the top level to reflect the vision and lessons learned from other members and expand vertically to incorporate additional levels of detail.

Current work efforts of the D2.24 working group is focused on the exchange of data between functions based on functional use case scenarios. It is also expected that the output of this working group will generate new candidates for standardization. Examples include standard data sets, display formats, business services and technology services. The related outputs of this effort will then be submitted to IEC TC 57 for consideration and adoption as a starting point for new IEC Standards.

1.5 Relationship to Other Standardization Initiatives

This document is not intended as a substitute for the standards developed under various international initiatives such as IEC working group TC-57 but is intended to align with those standards.

These architectural requirements also captures and is consistent with the work previously performed by the EAS Project (Enterprise Architecture Standardization), sponsored by the IRC (ISO/RTO Council). This North American Initiative, which has as primary objective to reduce the IT costs associated with ISOs and RTOs.

Within the energy and information technology industries, formal standards bodies, industry consortia, and ad-hoc working groups have developed architectural guidelines and supporting standards. The products of these efforts contain a significant body of knowledge and best practices that have been assembled by industry experts across the world. These architectural requirements have drawn of this body of knowledge and have utilized these existing standards and guidelines as was feasible.

In particular, the following guidelines and standards were utilized due to their complimentary mission, goals, and objectives.

- IEC 61968 [IEC61986-1]
- IEC 61970: Common Information Model (CIM)
- IEC 62325 CIM Market Extension
- IEC 60870-5-104: Transmission Protocols
- IEC 60870-TASE.2: Inter Control Center Protocols (ICCP)
- IEC 61850: Communication Networks and Systems in Substations
- W3C Web Services Architecture [W3CWSA] Service Oriented Architecture taxonomy, ontology, concept model and related standards

1.5.1 IEC 61968

The IEC 61968 series is intended to facilitate inter-application integration, as opposed to intra-application integration, of the various distributed software application systems supporting the management of utility electrical distribution networks. IEC 61968 is relevant to loosely coupled applications with more heterogeneity in languages, operating systems, protocols and management tools. IEC 61968 is intended to support applications that need to exchange data on an event driven basis. IEC 61968 is intended to be implemented with middleware services that broker messages among applications, and will complement, but not replace utility data warehouses, database gateways, and operational stores.

1.5.2 IEC 61970: Common Information Model (CIM)

The IEC 61970 standard created by TC57/WG13 and contains two main parts:

- Information models referred to as the Common Information Model (CIM)

- Application Program Interfaces referred to as Component Interface Specifications (CIS)

The CIM is described by a UML model that covers not only IEC 61970 but other applications domains described by IEC specifications, e.g. IEC 61968 from TC57/WG14 and IEC 62325 from TC57/WG16. The IEC 61970 part of the CIM is documented in the 300 series documents, e.g. IEC 61970-301.

- The CIS specifications define the following type of interfaces

- Platform independent API specifications described by the 400 to 449 series documents.

- A subset, i.e. profile, of the CIM used in a specific data exchange or context described by the 450 to 499 series documents.

- Platform specific API specifications where the 400 series documents are mapped to an implementation technology.

1.5.3 IEC 62325 CIM Market Extension

The IEC 61970 standard addresses wholesale market interactions. TC57 - WG16 is extending the current CIM market extension that was primarily developed from a North American perspective to take into account the European market design including the Over the counter market as well as integrating the European market standard defined by ETSO works into IEC.

1.5.4 IEC 60870-5-104: Transmission Protocols

IEC 60870-5 defines an RTU communication protocol. It should be noted that DNP is considered a derivative of this protocol.

IEC 60870-5 consists of the following parts, under the general title Telecontrol equipment and systems:

- Part 5: Transmission protocols - Section 1: Transmission frame formats

- Part 5: Transmission protocols - Section 2: Link transmission procedures

- Part 5: Transmission protocols - Section 3: General structure of application data

- Part 5: Transmission protocols - Section 4: Definition and coding of application information elements

- Part 5: Transmission protocols - Section 5: Basic application functions

- Part 5-6: Guidelines for conformance testing for the IEC 60870-5 companion standards

- Part 5-101: Transmission protocols - Companion standard for basic Telecontrol tasks

- Part 5: Transmission protocols - Section 102: Companion standard for the transmission of integrated totals in electric power systems

- Part 5-103: Transmission protocols - Companion standard for the informative interface of protection equipment

- Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles

1.5.5 IEC 60870-TASE.2: Inter Control Center Protocols (ICCP)

Inter-utility real-time data exchange has become critical to the operation of inter-connected systems within the electric power utility industry. The ability to exchange power system data with boundary control areas and beyond provides visibility for disturbance detection and reconstruction, improved modeling capability and enhanced operation through future security control centers or independent system operators.

Historically, utilities have relied on in-house or proprietary, non-IS standard protocols such as those developed by the Western Systems Coordinating Council (WSCC), ELCOM, and the Inter-utility Data Exchange Consortium (IDEC) to exchange real-time data. TASE.2 began as an effort by power utilities, several major data exchange protocol support groups for the protocols mentioned above, EPRI, consultants and a number of SCADA/EMS and protocol vendors to develop a comprehensive, international standard for real-time data exchange within the electric power utilities industry.

IEC 60870 was developed based on a multi-standard approach to allow (1) a quick implementation to meet European Common Market requirements by 1992 and (2) also allow long term development of a more comprehensive protocol. The first protocol was designated TASE.1 (Tele-control Application Service Element-1). The second protocol, based on TASE.2 over MMS, was designated TASE.2.

Successful first implementations of TASE.2 between SCADA/EMS control centers led to further expansion to allow communications between control centers and power plants. This expansion did not impact the basic services, but did lead to the development of specific power plant objects. These objects were incorporated into TASE.2. Similarly, protection event data objects were also added to support substation communications. The second edition of the TASE.2 standards also includes one new object for the exchange of general data reports and another new object for sending acknowledgements of complex data objects. TASE.2 (ICCP) has now been adopted in Europe as the standard approach as well. The User Guide (505) has two annexes describing connection to the CIM and remotely updating definitions and security. Also note that the OSI profile is no longer in general use.

1.5.6 IEC 62351: Data and Communications Security

IEC 62351 is information security for power system control operations. The primary objective is to “Standardize security of the communication protocols defined by the IEC TC 57, specifically the IEC 60870-5 series, the IEC 60870-6 series, the IEC 61850 series, the IEC 61970 series, and the IEC 61968 series”.

1.5.7 IEC 61850: Communication Networks and Systems in Substations

IEC 61850 defines a layered utility communication architecture. This architecture has been chosen to provide abstract definitions of classes and services such that the definitions are independent of specific protocol stacks, implementations, and operating systems.

The IEC 61850 series is intended to provide interoperability between varieties of devices. Communication between these devices is achieved by the definition of a hierarchical class model and services provided by these classes.

IEC 61850:

- Supports all substation automation functions comprising control, protection and monitoring
- Future-proof Architecture, easy extension and safeguarding of investments due to semantic data objects
- Provides engineering and maintenance support by means of the Substation Configuration Language based on XML
- Uses readily available industrial Ethernet and communication components

1.5.8 World Wide Web Consortium Web Service Architecture

The Web Services Architecture (WSA) provides a conceptual model and a context for understanding web services and the relationships between the components of this model. The architecture does not attempt to specify how web services are implemented, and imposes no restriction on how web services might be combined. The WSA describes both the minimal characteristics that are common to all web services, and a number of characteristics that are needed by many, but not all, web services. The WSA is an interoperability architecture: it identifies those global elements of the global web services network that are required in order to ensure interoperability between web services.

1.6 Web Services Clarification

Throughout this document web services terms are used when referring to service oriented aspects of this architecture. These terms are used to convey a clear and accurate understanding of Service Oriented concepts and semantics as defined by W3C, a recognized authoritative source of Service Oriented Architecture. These terms are not intended to imply the use of any specific W3C technology when implementing services. It is not necessary for implementers to use any specific W3C web services technology to be consistent with these architectural requirements. The Web Services Description Language (WSDL) is used herein as a formal expressive language to accurately document service interfaces.

2. System Architectural Objectives

The architectural requirements specified in this document are expected to have positive impacts on industry stakeholders in the areas described in the following subsections.

2.1 Lower Total Cost of Ownership (TCO)

The objective is to lower both the system supplier and the end user total development and support costs. These include:

- Platform development and maintenance
- Project implementation costs
- Multi-vendor integration costs
- Database and model development and maintenance
- Display development and maintenance
- System support and maintenance
- Migration costs
- Critical skill requirements and costs

2.2 Improved Business Continuity

This includes minimizing the impact of a planned or unplanned, partial or complete outage of grid operations including cyber attacks.

2.3 More Effective User Interface

This includes the ability for users to interact with the system through a common, role-based, customizable task oriented user interface regardless of the source of the underlying applications or data base. This will reduce or eliminate the reliance on multiple application specific user interfaces.

2.4 Improved Standardization

This includes broad compliance with developing industry standards, such as IEC TC 57, as well as general IT standards with increased technical compatibility with other systems.

2.5 Reduced Dependence on Proprietary Platforms

This includes the protection of the end-user's investment against major changes in the supporting technologies such as hardware and software platforms and to provide flexibility in selecting vendor solutions.

2.6 Higher Level of Interoperability

This includes the ability to efficiently integrate multi-vendor solutions, and to support the integration of third-party applications, including in-house developed applications. This also includes the integration of EMS/MMS Systems with other enterprise applications.

2.7 Increased Reuse of Software Components

This includes the ability to share common components in multi-vendor solutions, as well as support a more flexible integration environment such as upgrading one application component without the need to upgrade the total system.

3. System Architectural Qualities

System qualities describe technical behaviors that support the achievement of the objectives described in the previous section. These qualities are neither easily measurable nor testable before the system is deployed and operated in the user environment. They represent technical goals that the architectural requirements of this document are intended to provide.

3.1 Strategic Qualities

Strategic qualities reflect how well a solution meets technical business requirements. Strategic qualities are the primary criteria used to evaluate if an IT solution is judged to be "well-designed."

3.1.1 Scalability

Scalability is the ability of the system to support expansion in response to increased load on the system not related to changes in the functions that the system provides. It is important that the cost of scaling be minimized and be relatively linear as a function of load growth.

Load increases can take the form of an increased data volume, workload, network traffic, and/or number of connected users. The scalable solution is changed to respond to this load increase through the expansion, upgrade, and/or enhancement the baseline component configurations. Scalability has multiple dimensions, as solutions can grow by adding more servers or CPUs, increasing bandwidth, increasing memory, or adding more disk storage. Different approaches to scaling affects other system qualities such as Manageability. For example, adding a new server could reduce Manageability dramatically, while increasing the number of CPUs per server would have little impact on Manageability.

Scalability is the most important strategic quality. If reaching a growth business objective requires throwing away an existing solution, then that solution isn't scalable.

3.1.2 Flexibility

Flexibility is the extent to which a solution can adapt as business requirements change. It is also the ability of the same solution to be effectively used for multiple purposes. A flexible architecture facilitates greater reusability and faster deployment. However, the more flexibility a solution has the less that solution tends to map into the other system qualities. For example, scalable solutions tend to derive their scalability from anticipating growth along a particular path, but this approach diminishes flexibility. Flexibility also comes at a cost and may impact system performance. Therefore, the balance between flexibility, performance and cost is a major consideration.

3.1.3 Openness

Openness is the degree to which the configuration permits transparent interoperability and integration among multi-vendor applications. An open environment permits vendor-independent solutions, facilitates integration of third-party or in-house developed applications, and allows independence and flexibility. The downside of openness is cost of the solution itself, which needs to be weighed against the savings in the areas listed above.

3.1.4 Platform Independence

Platform Independence is closely related to openness. Software that is platform independent does not rely on special features of any single platform or handles those special features in a transparent way.

Key aspects of platform independence include:

- Operating System Independence
- Middleware Independence
- Database Management System Independence

If one visualizes openness in the “horizontal” direction, then platform independence is openness in the “vertical” direction.

3.1.5 Implementability

The ease with which configurations and applications can be deployed is related to the required level of effort, skills, and project duration. A target date and budget for a product’s deployment can be met with greater certainty with a product designed with ease of implementation in mind.

3.2 Systemic

The solution’s operations depend on systemic qualities. These constraint-oriented qualities closely relate to solution complexity. Besides those described below, other systemic qualities include reproducibility (backups and restores), documentability (useable documentation) and testability.

3.2.1 Maintainability

Maintainability is the set of attributes that bear on the effort needed to make specified modifications[1] and is closely related to the system TCO. This includes the degree to which system administration resources can maintain the system including its software, hardware, and documentation components. It is the degree to which the configuration minimizes maintenance requirements, supports problem isolation and supports easily replaceable components.

3.2.2 Manageability

Manageability is the degree to which application management resources can operate the system. Manageability addresses the ease with which IT professionals can handle the solution on a tactical day-to-day level.

---

[1] ISO 9126

Managing a vertically scalable solution, such as a single box with 32 processors, is usually easier than managing a horizontally scalable solution, such as 32 one-processor boxes. Of course, the failure of one box in a distributed 32-box solution is easier to manage than the failure of the single 32-processor box. The trade-off in this case is availability.

Manageability also depends on the human resources needed to operate the solution. As a rule, commercial-off-the-shelf (COTS) solutions are more manageable than custom-built solutions. Where possible, COTS solutions should be investigated and deployed to the extent these products are available to reduce internal development or support costs and reduce implementation risks.

3.2.3 Securability

Securability is similar to manageability in the way it affects the solution. Securability includes the following features:

- **Authentication** – Establishment of the validity of the originator of a transmission or message.
- **Authorization** – Verification of an individual's right to access specific system resources.
- **Availability** – Assurance that allowing the sharing of information will not create situations in which authorized users are denied service.
- **Confidentiality** – Assurance that information is not disclosed to unauthorized persons, processes, or devices.
- **Data Integrity** – Assurance that data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.
- **Non-repudiation** – Assurance that the sender of data is provided with proof of delivery and that the recipient is provided with proof of the sender's identity, so neither can later deny having processed the data.

As the solution's complexity increases, maintaining security becomes increasingly difficult. Horizontally scalable solutions have more potential security holes, but the simplicity of a horizontally scaled solution may provide a more manageable security environment. A very large, multi-vendor, business-to-business solution will be difficult to manage and difficult to secure. However, if common architectural principles are developed and supported by multiple vendors, this security risk can be mitigated.

Secure systems can be built, but they require definition and enforcement of company-wide security policies and procedures that support capabilities such as security zones, personnel categories, personnel category access to security zones, and assignment of individuals to personnel categories.

3.3 Service-Related

Service-related system qualities address runtime systems level issues that affect service.

3.3.1 Availability

Availability is the percentage of the scheduled uptime during which a system is actually available to users. Availability affects service at many levels.

There is a tendency to equate availability with IT infrastructure components. A broader definition addresses the broader problem of whether the user can reliably perform all the needed work with of the solution. For example, system availability would be limited if a scheduling coordinator could view historic data reports but could not submit a new schedule.

3.3.2 Business Continuity

Closely related to availability, business continuity is the ability to eliminate or recover from outages of system components within specified time frames via the use of redundant back-up and off-site capabilities.

An organization's Business Continuity requirements are based on the impact of system not being available and the cost of providing business continuity capabilities. For example, an ultimate Business Continuity scenario may require uninterrupted system operations, including the case of physical loss of the primary control center even though this solution is a very costly option.

3.3.3 Performance

Performance is a critical measure of system quality. Typical architectures define performance goals and strive to balance the business/costs requirements with the system qualities to achieve performance objectives.

3.3.4 Efficiency

Efficiency is a set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

3.4 User-Related

The most important user–related system qualities are usability and accessibility.

3.4.1 Usability

The measure of usability is the effectiveness of users attempting to complete a business procedure. This includes how well the human-machine interface is integrated with the business functions and activities of the user including the availability of specific business functions. Outcomes such as an intuitive user interface and consistent look and feel are common desirable usability traits.

3.4.2 Accessibility

Accessibility is the degree to which the architecture permits controlled access to data, uses standard communication interfaces, and uses non-proprietary databases. The concept of

accessibility is maturing to embrace making solutions accessible to users anywhere at any time. Obviously, engineering for greater accessibility increases system complexity and affects manageability, security, and scalability. Whereas availability is influenced by the solution infrastructure, accessibility describes the user's modes for accessing the solution.

4. System Architectural Principles (Design Tenets)

While the system qualities defined above are directly observable for currently deployed applications, they are not easily measurable or testable for applications before they have been deployed in a user's environment. Systems qualities are by-products of the application of system architectural principles or design tenets. However, it is important to note that some design tenets could be in conflict with system qualities (e.g., a highly modular, abstracted, encapsulated, and loosely-coupled system implemented with message oriented web services may easily run into scalability, performance, and availability problems). It should be recognized that the more a solution employs these design tenants the more likely that solution is to exhibit the system qualities that are of strategic importance resulting in a more open and flexible solution.

The design tenets provided below serve as goals and objectives for the architectural requirements provided in sections 7 through 10 of this document. The intention has been to specify architectural requirements that exemplify these design tenets but are consistent with achieving the system qualities described in the previous section.

Table 1: Design Tenets

	Tenet	Short Name
1	Services are implemented as modular components with business-significant granularity	Modular
2	Services are configurable	Configurable
3	Services are customizable	Customizable
4	Services are open	Open
5	Services interact abstractly with other services	Abstract
6	Services are loosely coupled	Separate
7	Services are technologically neutral	Neutral
8	Services are secure	Secure
9	Services provide encapsulation	Encapsulated
10	Services provide unique non-trivial functionality	Unique
11	Services are instrumented	Instrumented

4.1 Tenet 1: Modular

Services are implemented as modular components with business-significant functionality and granularity.

A modular solution provides modules with well-defined functionality and interfaces. Modularity encapsulates all of the principles of Component-Based Development (CBD) and Service-Oriented Environments (SOEs), two key features of modern application architecture. CBD principles may be implemented in various technologies, but at the heart of CBD is the notion that if business services are designed as components, they are inherently reusable (as opposed to being designed for obsolescence).

A component is defined as an identifiable piece of software that describes and/or delivers a set of meaningful services that is only used via well-defined and documented interfaces or as an independently operable unit that is a part of the total structure. Object-Oriented (OO) techniques, which are the basis of the many modern application development methodologies, are based on defining application services, components, and modules called objects.

4.1.1 Benefits

Modularized services:

- Provide a well-defined function and interface. A module also has a well-defined usage of other service interfaces,
- Are inherently and explicitly designed to adapt to change;
- Are functionally partitioned into discrete, scalable, reusable modules consisting of isolated, self-contained functional elements;
- Rigorously use disciplined definitions of modular interfaces, including object-oriented descriptions of module functionality; and
- Are designed for ease of change, to achieve technological transparency, and, to the extent possible, make use of common industry standards for key interfaces.

4.1.2 Mechanisms

Business Process Modeling (BPM) is typically used to identify the modules or services related to a functional domain. Frameworks (such as Microsoft's .NET or OMG CORBA) and standards (such as J2EE and Web Services Descriptive Language [WSDL]) are often used to define services, modules, and components. Interface Definition Languages (IDLs, such as WSDL) are used to publish the services' interfaces, while interface repositories (such as Universal Description, Discovery and Integration [UDDI]) are used to share the interfaces for dynamic resolution.

4.2 Tenet 2: Configurable

Services are configurable.

Configurability describes the ability to easily reconfigure the component or service. Configurable components may be run in numerous physical topologies and be invoked in a number of manners. For example, a service will typically surface parameters related to how it connects to a database.

Configurable solutions also tend to surface constants associated with the business logic in a manner that lets them be easily modified. For example, a service that applies a price cap might allow the numeric value of the price cap to be modified.

4.2.1 Benefits

Configurable services can be reused and quickly modified as physical and business topologies change, and have clearly documented configurable parameters.

4.2.2 Mechanisms

Configuration files, deployment descriptors, run-time parameters, and table-initialized parameters are all common mechanisms for providing configurable services. However, it is important to note that the specific mechanism for providing configurable services is not nearly as important as the number of configurable attributes that a service possesses.

4.3 Tenet 3: Customizable

Services are customizable.

Applications must also be capable of being customized to business requirements. In the past, packaged applications often made a virtue of requiring that the business align its functions with the package. Today, this is recognized as inappropriate and usually impossible, because of the timescale of business change. Although numerous companies might use similar services, there will always be the need to implement business logic according to individual specifications.

4.3.1 Benefits

Customizability allows a service to be quickly adapted as the business changes and reused in a wide variety of situations.

4.3.2 Mechanisms

Inference engines, scripting languages, Object-Oriented Programming (OOP), and extensible frameworks are all mechanisms for allowing customization.

4.4 Tenet 4: Open

Services are open.

An open system is a collection of interacting software, hardware, and human-interface components that:

- Are designed to satisfy stated needs; functionality requirements
- Have component interface specifications that are:
    - Fully-defined,
    - Available to the public, and
    - Defined in documents and/or models (e.g., UML)
- Are defined by forums acknowledged to develop standards specifications that are:
    - Available to the public as a standard, preferably at an international level or from an industry level standards body (e.g., OMG, OPC-foundation etc)
    - Maintained through special interest group consensus; and available to all group members. For such specifications to be open, the interest group must make the specifications publicly available
- Are implemented such that the components conform to the interface specifications.

4.4.1 Benefits

A systems architecture that is based on open systems allows programs to leverage commercially funded or developed technologies and thereby permit the users to take increased advantage of competition. It allows quicker implementation of superior new systems and faster upgrades of legacy systems with less complexity and at lower cost.

4.4.2 Mechanisms

Open systems leverage existing standards, documentation and published interfaces, and provide compatibility test suites to demonstrate that the implementations conform to the documented interfaces.

4.5 Tenet 5: Abstract

Services interact abstractly with other services.

Abstraction is “the expression of a quality apart from a particular object or specific embodiment.” Abstraction is related to encapsulation; it is a mechanism for reducing complexity and increasing efficiency.

Abstraction also tends to have the effect of reducing cohesion between services. Abstraction will often define a simplified interface that wraps a much more complex set of interfaces. For example, a complex set of relational tables in an RDBMS might be surfaced using a view; or the functionality of a messaging product might be surfaced using an abstracted interface. Abstractions allow a service to leverage other services in a simplified manner while reducing cohesion.

4.5.1 Benefits

Abstract services have:

- Reduced complexity;
- Reduced cohesion or coupling; and
- Simplified interfaces.

4.5.2 Mechanisms

Abstraction is implemented by generalizing functionality through techniques such as Object-Oriented Analysis and Design (OOAD) and the use of design patterns that introduce abstraction, such as mediators, Model-View-Controller (MVC), and views.

4.6 Tenet 6: Separate

Services are loosely coupled.

A loosely coupled system is one that reduces the dependencies between services. These dependencies include, but are not limited to, transactions, security, conversational state, and location. Minimizing the context information that is shared between them, the more loosely coupled are the services, increases the amount of separation achieved among the services.

4.6.1 Benefits

The benefits of loosely coupled systems include the ability to change the:

- Locations of services;
- Names of services;
- Identities of services;
- Definition or structure of services;
- Variety of services;
- Amount (number) of services;
- Size of services; and
- Timing of activities and processes across services.

4.6.2 Mechanisms

The mechanism for integrating services into business applications is typically a messaging technology. Message-based architectures also provide the ability to provide loose coupling across time through asynchronous paradigms as well as store and forward mechanisms.

Message-oriented architectures are commonly used to produce loosely coupled services. A message-oriented architecture that is based on standard message content allows the services to be loosely coupled, be asynchronous (with inherent advantages for exploiting parallelism), and exert far better control over the business process. Message-oriented architectures define services that help expose messaging interfaces that may be invoked either synchronously or asynchronously. The message content, however, can create an implicit coupling between the services, resulting in potentially significant software changes if the services or message content change. Therefore, control of the message content is important and standardized content definitions will help achieve the loose coupling.

4.7 Tenet 7: Neutral

Services are technologically neutral.

Services that are technologically neutral do not favor a specific platform or invocation mechanism.

4.7.1 Benefits

Neutral services:

- Offer more opportunity for reuse;
- Can be deployed in a wider variety of environments and topologies; and
- Do not dictate a specific language for invoking the service.

4.7.2 Mechanisms

Neutrality is typically accomplished through:

- Standard or widely adopted protocols, such as eXtensible Markup Language (XML) and Transmission Control Protocol/Internet Protocol (TCP/IP);
- Platform independent implementations, such as Java and American National Standards Institute (ANSI) C (however the use of specific platform software libraries must be avoided to maintain portability) ; and
- Use of components that support multiple, widely available platforms, such as MOM products with implementations on multiple platforms.

4.8 Tenet 8: Encapsulated

Services provide encapsulation.

Encapsulation is the gathering of related pieces of data together with the operations performed on that data. The essential characteristic of a service is this grouping of data and methods (operations) into a "black box" that only surfaces the service's business functionality. Thus, the interface to the service provides access to its business logic without the necessity for understanding the internals of the implementation. For example, it is irrelevant whether a data store is implemented in a database or in memory.

The key to successful encapsulation is to only access the service via well-defined interfaces. The service should be responsible for managing its data and implementation.

Services that manipulate large amounts of data increasingly rely on an external service such as a Relational Database Management System (RDBMS) for persisting and loading data. It is important that these services provide clean separation of data and functionality, so that they can be managed as self-contained entities. This can be accomplished by treating the service providing storage (RDBMS) as any other service and accessing the data via well-defined interfaces. In effect, these interfaces become data services.

4.8.1 Benefits

Encapsulation is the key to providing effective services, and services are the keys to supporting adaptive business processes. Encapsulation also allows services to be easily replaced by new components offering the new services.

4.8.2 Mechanisms

The primary hallmark of encapsulation is the definition of immutable interfaces for the service. These interfaces can be surfaced via repositories or across message-based architectures, but it is important that the interface actually surface functionality of business significance while protecting the data and the implementation that produces the functionality.

4.9 Tenet 9: Secure

Services are secure.

Security is about controlling access to a variety of resources, such as application components, data, and hardware. There are four concepts upon which security measures are based:

- **Authentication** – establishment of the identity and authenticity of the party with whom applications communicate;
- **Authorization** or Integrity (Access Control) – Use the authenticated identity of the communicating party to determine access rights;
- **Confidentiality** or Data Protection – provision of data privacy, integrity, and non-repudiability; and

- **Auditing** – logging and monitoring of events that occur in a system that are related to security.

The security of an application is impacted by numerous design choices, such as the selection of communication protocols and the method of user authentication. Most security vulnerabilities are not in security-related code. Instead, they are found in code that is unrelated to security and that was written without attention to security. This is why developers must be keenly aware of application security requirements.

4.9.1 Benefits

The benefits of security include:

- Data privacy;
- System reliability;
- System availability;
- System integrity;
- Non-repudiation;
- Organizational reputation; and
- User confidence
- Fulfillment of regulatory obligations (e.g., NERC, SOX)

4.9.2 Mechanisms

There are numerous mechanisms for providing security, including:

- Public Key Infrastructures;
- Lightweight Directory Access Protocol (LDAP) repositories;
- Infrastructure security services via XML;
- Federated security services; and
- Scanning services for weaknesses or exploits.

4.10 Tenet 10: Unique

Services provide unique functionality.

Unique non-trivial services provide functionality that is not available from other services. Services should rely on other services as applicable for providing needed functionality. Common

examples are services that in turn use the services of an RDBMS, LDAP, or Domain Name Server (DNS).

Services should also be designed with an understanding that they can and will be reused in any number of unforeseen manners. However, there is a caveat: if other services are not used appropriately, the inverse affect of increasing cost might result. For example, reusing the services of an RDBMS for storing non-keyed (flat) files is not appropriate, since the purpose of the RDBMS service is to provide optimized and transactionally safe data storage and retrieval: it therefore makes sense when the data are uniquely identified through keys.

It can also make good sense to use a RDBMS for data other than keyed data structures, such as to provide easy access to third-party user interface and reporting tools, to provide a transactionally safe environment with guaranteed data consistency through locking capabilities that support multiple user access, and to provide an easy interface to COTS products, which typically support data retrieval from an RDBMS while enabling additional services and advantages (e.g., uniformity of data persistence in a component or solution), high availability of data, data backup solutions, etc.

4.10.1 Benefits

The benefits of providing unique functionality include:

- Reduced code base;
- Reduced component count; and
- Reduced functional overlap (one and only one implementation of a business function).

4.10.2 Mechanisms

Uniqueness and appropriate utilization of other services is often ensured by:

- Leveraging other services as appropriate;
- Using repositories for discovering available services;
- Clear documentation on the dependency of the service on other services; and
- Clear documentation defining the functionality a service provides.

4.11 Tenet 11: Instrumented

Services are instrumented.

Services must be instrumented to be globally manageable in order to provide reliable and maintainable solutions. As critical business functionality is accomplished by using services, it is imperative that the services be proactively managed, just as hardware and network infrastructure are typically managed. As services are reused across system boundaries, this management functionality must also span system support groups.

Instrumentation enables the state and performance-related metrics associated with a service to be monitored and allows services to be managed (started, stopped, deployed to different environments, etc.) remotely. The more thoroughly instrumented a service is the more reliable and scalable it becomes.

4.11.1 Benefits

The benefits of instrumentation include:

- Dynamic scalability (the ability to react to load conditions dynamically, thus consuming resources only while serving load);
- Increased reliability; and
- Reduced cost (by generalizing monitoring functionality).

4.11.2 Mechanisms

Services are instrumented through implementation of management standards and surfacing management-related interfaces such as JMX, JSR-77, SNMP, etc. In addition, services must be designed to be uniquely identifiable – especially when they are assembled with or use other services.

Instrumentation can be provided for in many ways and not necessarily with a single tool. Monitoring of a service can be passive or active, or services might be self-correcting. The service must be instrumented in such a manner that it is able to surface relevant information within required timeframes.

5. System Performance Considerations

Performance is critical since even though a system may process a function correctly if a function is not completed in an acceptable time the result may be of little to no value. Failure to meet acceptable performance may significantly diminish or eliminated the expected system benefits. Specific performance requirements should be based on expected operator and end user business requirements and cannot be generalized. This intent of this section is to present suggestions for performance criteria, metrics, methodology, and tools. However, specific test plans need to be based on the technical characteristics of the application and business criteria for expected volumes as well as operating scenarios and events.

5.1 Scope

Performance testing does not include functional testing and assumes valid test data. The goals of performance testing are to:

- Validate performance under various load profiles and identify areas where performance results exceeded the established limit criteria;
- Reveal bottlenecks and breaking points;
- Verify the impact of non-responsive systems on overall performance results during testing cycles; and
- Identify non-optimal system settings.

Performance tests should simulate major transactions and test all technology components of the system architecture including web layer, middleware, core system, database, interfaces and other third-party systems. All external and internal user-facing systems and communications with all internal systems integrated with the system under test are in scope. This includes validation of user and data load scenarios based on real business-day operational schedules.

Performance testing addresses performance and load validation of the integrated system, treating each individual component as a “black box.” The testing uses performance criteria for each component based on the results of standalone performance testing.

5.2 Base Test Conditions

For each performance scenario, the following base case conditions should be enabled:

- The system should be configured with all hardware and functions as production.
- All system functions should execute at the periodicities and execution times consistent with normal operations.
- Each monitor at all consoles (including operations, support, and management consoles) should present all “common information” deemed to be part of the normal display arrangement.

- Data should be retrieved and sent via the interfaces to external computing environments.

5.3 Performance Scenarios

System performance should be based on detailed loading parameters for normal loading and high activity loading scenarios. Separate system performance criteria should be established for each scenario. The primary objective of system performance testing and associated loading parameters is to ensure the system is properly sized to ensure minimum levels of performance during normal activity and high activity including future system expansion.

Specific criteria should be developed for both scenarios to represent expected system activities that would be expected in those states. The system activity for the high activity state should approximate the 'worst case scenario' based on volume due to abnormal business conditions, events and business growth. Alarm rates and data changes should be consistent with this level event.

5.3.1 Normal Scenario

The normal scenario consists of the base case scenario with typical system loading volumes. The system should be configured as close as possible to actual field conditions. Typical operator and user actions should be simulated. Inputs such as data from the data acquisition subsystem should be active or simulated.

5.3.2 High Activity Scenario

The high activity scenario should consist of the base conditions and additional processing resulting from worst case events with the addition of business growth. The data acquisition subsystem should be stressed to approximate the actual increased load from rapidly changing inputs or field telemetry based on such a scenario. Operator and user activities that would be employed during an emergency event should be simulated.

5.3.3 Resource Utilization Metrics

The average utilization over the time of the test scenario should be calculated as the used capacity of the resource divided by the total available capacity of the resource. For example, processor average utilization may be calculated as busy time divided by total time. LAN average utilization may be calculated as the quantity of data transferred (Mbits) divided by the LAN data rate (Mbits/second) multiplied by total time (seconds). Note that this metric should be used in conjunction with user and application response metrics to ensure overall system performance levels are maintained across a given period of time.

The average resource utilization of each system resource for each activity scenario should not exceed a specified threshold.

- Average utilization of the processing capacity of any processor used for executing application functions should not exceed, for example, 35%-40% averaged over the test period;

- Average utilization of the transfer capacity of each storage device should not exceed, for example, 30%- 40%; and

- Average utilization of any LAN should not exceed, for example, 10%-25%.

5.4 On-line Transaction Based Load Profiles

Testing for on-line transaction based system components should be based on system performance under conditions that are close to production levels as well as stressed conditions. System behavior may be different during specific operational time periods or events. Optimization techniques used within the system may skew the results unless proper measures are taken. For instance, Event Handler Pooling might give rise to better results when the same type requests are continuously sent to the Enterprise Service Bus.

Testing should include multiple load profiles to simulate real-life load on the system. These are:

- **Baseline** (benchmark) profile
- **Workload** (projected) profile;
- **Stress** profile (expected and “hammer”);
- **Spike** profile; and
- **Scheduled Maintenance** profile.

The **Baseline** profile involves just a few users (3-8). The only purpose of the baseline profile is to verify test cases and scripts.

The **Workload** profile is based on business operations expectations. The model should simulate requests hitting the front end when testing end-to-end paths. It should also simulate virtual user transactions when testing internal systems. The workload profile is created with selected use cases that contribute most of the system load. A load profile defines the total number of users and their distribution across operations.

The **Stress** profile retains the distribution of the Workload profile; however, it typically assumes at least 100% more transaction or users. A “hammer” stress profile is used to deliberately force the system into a “not working” condition or breaking point.

The **Spike** profile has a load distribution that more closely simulates spike loads. It typically presumes up to a 200% increase in number of transactions or users for 10-20 minutes.

The **Scheduled Maintenance** profile is based on the Workload profile with resources typically reduced up to 50% for up to 24 hours.

5.4.1 On-line Performance Criteria

The performance criteria stated below should be based on the performance requirements defined in Service Level Agreements (SLAs) and Business Specification documents.

- The system should be stable and responsive in accordance with the projected workload over typical operational time periods. A system is considered to be stable if it is capable of returning to its normal state after transient spikes in load. No response time degradation should take place over the period.

- The system should exhibit the defined response times typically 99% of the time under the projected workload.

- The system should sustain the SLA-defined response limit over a typical workday.

- The system should be responsive under the various work load scenarios that have been selected.

- The system should be stable both during spike period and during the transition to a normal working mode. The spike load represents typically at least 100% of the projected load within a short period of time. Intermittent peak volumes of doubled the number of transactions per second over a defined period should not impact response times by typically not more than 125% of delay limits.

- The system should be responsive under the stress load or an atypical high load. The system should be stable both during the stress period and during the transition to the normal working load mode. An extreme variation of the stress load is sometimes used to determine the breaking point of the system.

- The system should be stable with no response time degradation under the projected workload during the scheduled maintenance period.

5.4.2 On-line Performance Metrics

The metrics below can be used to measure performance of on-line real-time systems such as web based applications.

5.4.2.1 Response Time

The weighted response time $\mathbf{T_w}$ is a linear combination across the tests:

- $\mathbf{T_w = \Sigma\ t_i * a_i}$ (where $\mathbf{\Sigma\ a_i = 1.0}$) and the maximum response time $\mathbf{T_{max} = MAX(t_i)}$, where:

- $\mathbf{t_i}$ is the response time for the **i**-th test in a full run and

- $\mathbf{a_i}$ reflects a summary of all types of transaction requests.

5.4.2.2 Transaction Rate

The weighted transaction rate $\mathbf{TR_w}$ is a linear combination of different transaction requests across the tests:

- $\mathbf{TR_w = \Sigma\ tr_i * a_i}$ (where $\mathbf{\Sigma\ a_i = 1.0}$), where:

- $\mathbf{tr_i}$ is the number of transactions per second for the **i**-th test and

- $\mathbf{a_i}$ reflects a summary of all types of transaction requests.

5.4.2.3 Other Metrics

Additional measurements may be taken based on the above equations. The weighted measurements above provide averages with the maximum and minimum measurements as the extreme observations. If performance is not satisfactory, additional specific system counters can be recorded and analyzed. For example, other measurements could include web server hits per second, throughput, retries per second, total connections, and connections per second. Testing should span short operating time periods of 1-24 hours as well as long time periods of at least seven days.

6. System Architecture Guidelines

The architectural guidelines described herein embrace the characteristics and attributes that enable interoperability among applications from different solution providers. *Interoperability* "is the capability of systems or units to provide and receive services and information between each other, and to use the services and information exchanged to operate effectively together in predicable ways without significant user intervention."[2] The architecture endeavors to exhibit the characteristics and attributes of interoperability as defined in GridWise Architecture council's interoperability checklist including:

- Open architecture
- Technology neutrality
- Multiple vendor competition and innovation
- Open standards based
- Interface point specifications
- Mission critical redundancy
- Cyber-security

"Systems with high interoperability have lower equipments costs and lower transactional costs, higher productivity through automation, more conversion of data and information into insight, higher competition between equipment suppliers, and more innovation of both technology and applications." [2]

One of the key aspects of the architecture for the next generation of EMS/MMS systems is the use of Service Oriented Architecture (SOA) principles. SOA is based on the concept of independent building blocks (or "Components") communicating together via a standard "Integration Layer". Having these Components appear on the Integration Layer as "Services" is a standardized software implementation technique that allows Components to be shared among independent applications.

The architecture is comprised of the following sub-architectures:

- **Information Architecture** covers the enterprise data models and enterprise data standards. The former includes mappings to the Common Information Model descriptions (CIM) and the CIM for Market Extensions (CME), common CIM/CME

---

[2] Gridwise Architecture Council, Introduction to Interoperability and Decision-Maker's Interoperability Checklist, Version 1.0, April 2007

messaging standards, and repositories for CIM/CME-related information. The information architecture is divided into two parts according to the encapsulation tenet:

Configuration data describing properties inherent to objects related to the process (e.g., for a power system transmission lines - lengths, cable types etc.)

Application specific data related to particular applications (e.g., for power systems - the pi-equivalents, load flow slack bus etc.). Application specific data requires that the information model itself adhere to the application architecture.

- **Integration Architecture** describes the key design principles and key categories of components of the proposed architecture, including the integration layer, business service categories, and technology service categories. Although modularity is a fundamental design principle of a component-based architecture, the primary intent of this section is to describe how the Components are organized and interface together in a real-time or near real-time EMS/MMS System environment.

- **User Interface Architecture** describes the architectural requirements and applicable standards for a common, application-independent and database-independent User Interface for use by the central operator control room. It also includes requirements for model display and data maintenance tools.

- **Security & Network Architecture** defines the security standards and services including key areas of communications network segmentation, perimeter protection; system and communication protection; user security, centralized audit and monitoring; change management and system integrity, shared storage plus virtualization. The security architecture relies on a security policy that is based on standards, such as NERC CIP. The network architecture requirements are limited to cyber security issues and do not addresses typical network design and architecture issues.

- **Platform & Business Continuity Architecture** addresses not only platform convergence to more efficiently utilize the computing resources in a multi-vendor environment but also includes server and storage virtualization, capacity planning, configuration management, software deployment, and systems monitoring. The architecture also describes the framework for providing various levels of control center redundancy; describes the modes of operations; as well as the associated requirements for data synchronization, application configuration and source model maintenance.

7. Information Architecture

7.1 Overview

Information Architecture is a translation of the business constraints into a description of how information flows throughout the system. The concepts addressed in herein are fundamental to the EMS/MMS System architecture of the future and have a significant impact on other architectural areas of this document.

Key objectives of the information architecture are to:

- Facilitate interoperability of software components
- Achieve a model-driven design for data that allows for evolving data requirements.

These goals require a set of standard interface agreements by which business components exchange data. Typically, a business component requires some data produced by another component and produces data that is used by one or more other components. To achieve the desired open environment, components should be able to be developed and maintained by different vendors, so this exchanged data will commonly be a fundamental part of the interface between vendor products. The Information Architecture specifies how such data exchange agreements will be made.

7.2 Scope

The Information Architecture specifies how Data Sets are used to define standardized communication among the business services. The term “Data Set” is used as a neutral term to describe a module of data of specific purpose and semantic structure in a manner that is independent of the specific vehicle used to store it (e.g. files, databases) or transport it (e.g. message buses).

The scope of the Information Architecture includes the following:

- The Logical organization of information
- The methods by which the structures of Data Sets are defined.
- The standard interface mechanisms by which business components produce either full or incremental updates of Data Sets.
- The standard interface mechanisms by which business components access Data Sets or consume full or incremental updates.
- The standard interfaces between the information service layer and technical services, if any. (In the CIGRE D2.24 architecture, technical services are distinguished from business services in that a technical service is reused in many business contexts, such as logging or backup.)

The Information Architecture adopts the notion of a Canonical Data Model (CDM) from which standard Data Set specifications are derived. The Information Architecture does not, however, include a definition of either the CDM, or any specific Data Set specifications that will be derived from it. These are all presumed to be driven by the specific requirements for any particular business process and documented separately as a result of analyzing those requirements.

The Architecture inherits the generic terminology from IEC-61968, namely Business Functions, Business Sub-Functions, and Components. Additionally, the term Services has been chosen to represent the specific functions within a Component that handle data exchanges.

It is important to note that in Power System Modeling, the process of managing the "master model" specification for the electrical power system is considered to be a business process, and not part of the generic information architecture. The details of Power System Modeling are driven by the business process needs. Therefore the architectural premise is that there is no essential difference between the model update interface and any other interfaces among data producing and data consuming components. Some additional requirements include:

- The ability to build and maintain the power system model as a single centralized model, that is the source of the data for all current and future applications/implementations, including system upgrades or replacement

- Integrated power system modeling: The ability to build and maintain the power system model and displays in a fully integrated process, ensuring consistency between the data model and its graphic representation

- Integrated system modeler: This implies that there are two complementary and interlinked views of the power system model: a database centric view and a graphic view (One-line and/or geographic). It will be possible to build/maintain the model through either view and to navigate seamlessly between the views. The modeler will be capable to generate displays and model data compliant with the specified standards.

7.2.1 Business Function Decomposition (BFD)

To identify and define specific Data Sets, it is necessary to decompose business activities into Components that interact with one another by exchanging Data Sets. Components provide standardized, generic services that can be invoked (or consumed) by various independent applications. Components provide Services that support functions, such as providing the estimated state of the power system. Conversely, each Abstract Component can consume services provided by other Components, for example receiving a list of approved generation outages. The service definitions outline the details such as service type, interaction model, criticality, and dependencies when necessary. [See Section 7.7.3]

Just as the use of a Canonical Data Model is stipulated by the architecture, but the specific definition of the CDM is a separate exercise, the use of a Business Function Decomposition is stipulated, but the specific BFD is left to detailed analysis of business processes. A BFD should consist of Business Functions (at the highest level) that are further decomposed into Business Sub-Functions. A process for carrying out this decomposition is given in the next section.

An example of what a BFD for EMS/MMS Systems markets might look like is given in the below figure.

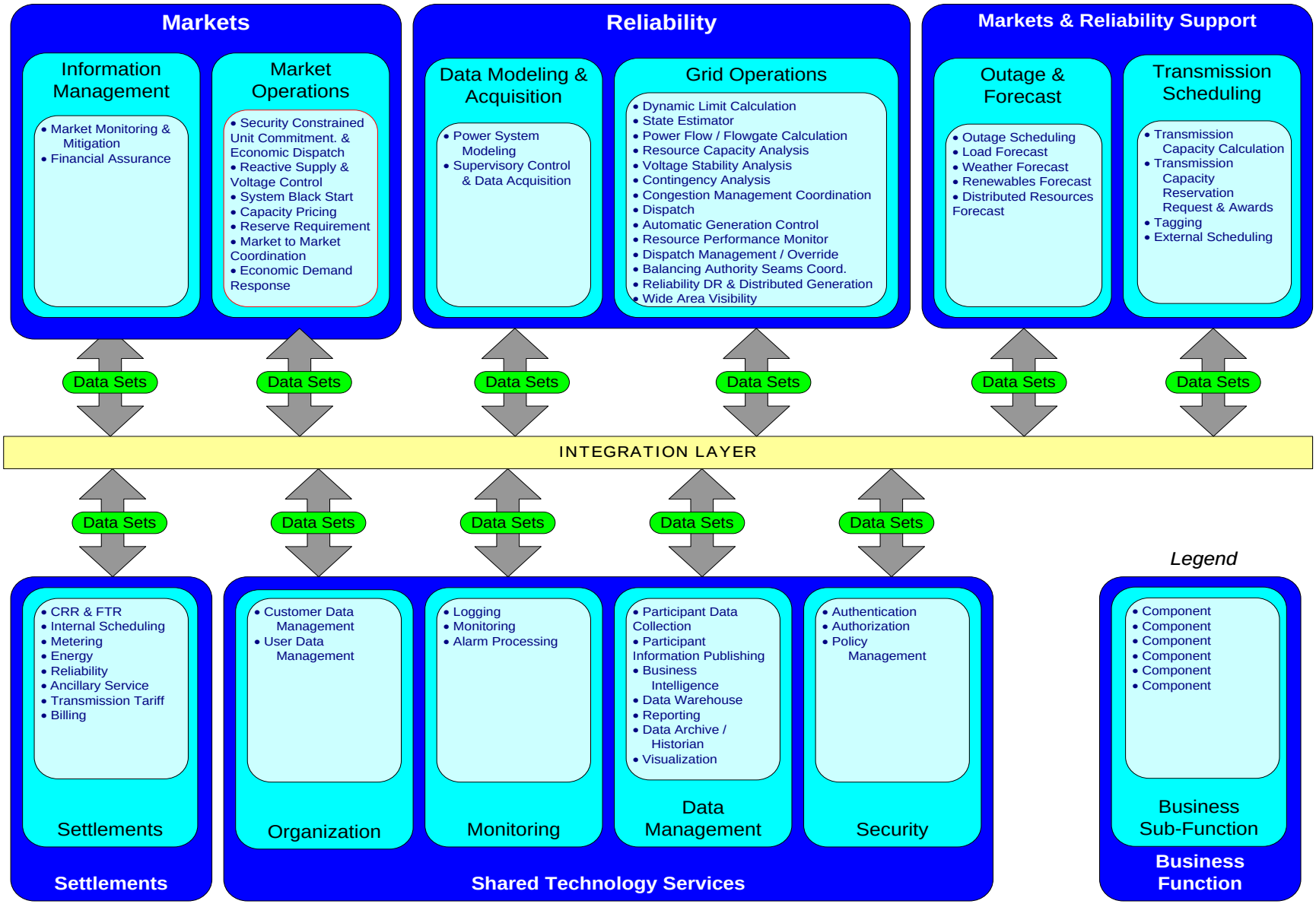

Figure 7-1: Example of Business Function Decomposition

In this example decomposition, the reasoning is that Electricity System Operators have two major responsibilities: to ensure electricity grid reliability and to operate efficient wholesale electric markets, where applicable. Therefore, the first two Business Functions are named ***Reliability*** and ***Markets***. The Reliability Business Function includes processes related to maintaining the power system model, collecting SCADA data, solving power flows, and dispatching resources, while the Markets Business Function covers the areas related to collecting bid/offer data, determining economic dispatch solutions, selecting ancillary service awards and so on.

Reliability & Markets Support is a third Business Function related to those areas needed by the two primary Business Functions, such as forecasting demand, tracking outages, and reserving and scheduling transmission. The last Business Function is ***Shared Services***, which provides technology-focused capabilities common to multiple areas, including central monitoring, data collection & reporting, security services and complex mathematical functions. In the future, other Business Functions may be added to the scope of the initiative, for example Settlements & Billing.

7.2.2 Service Design Process

The service design process is a top-down process with iterations. It starts with business process analysis that provides understanding of how the components interact with one another. Next, service identification is conducted based on the interactions and message flow. Detailed use case analysis is carried out for the details about service input, output, dependencies, and other characteristics. Domain data modeling is one of the important parts of the process, which captures data requirement with a formal technology so that business semantics can be represented in a consistent way. Given the complexity of the overall system, it is necessary to iterate the whole process until a good set of services is identified and defined. Figure 7-2 illustrates the overall service design process.

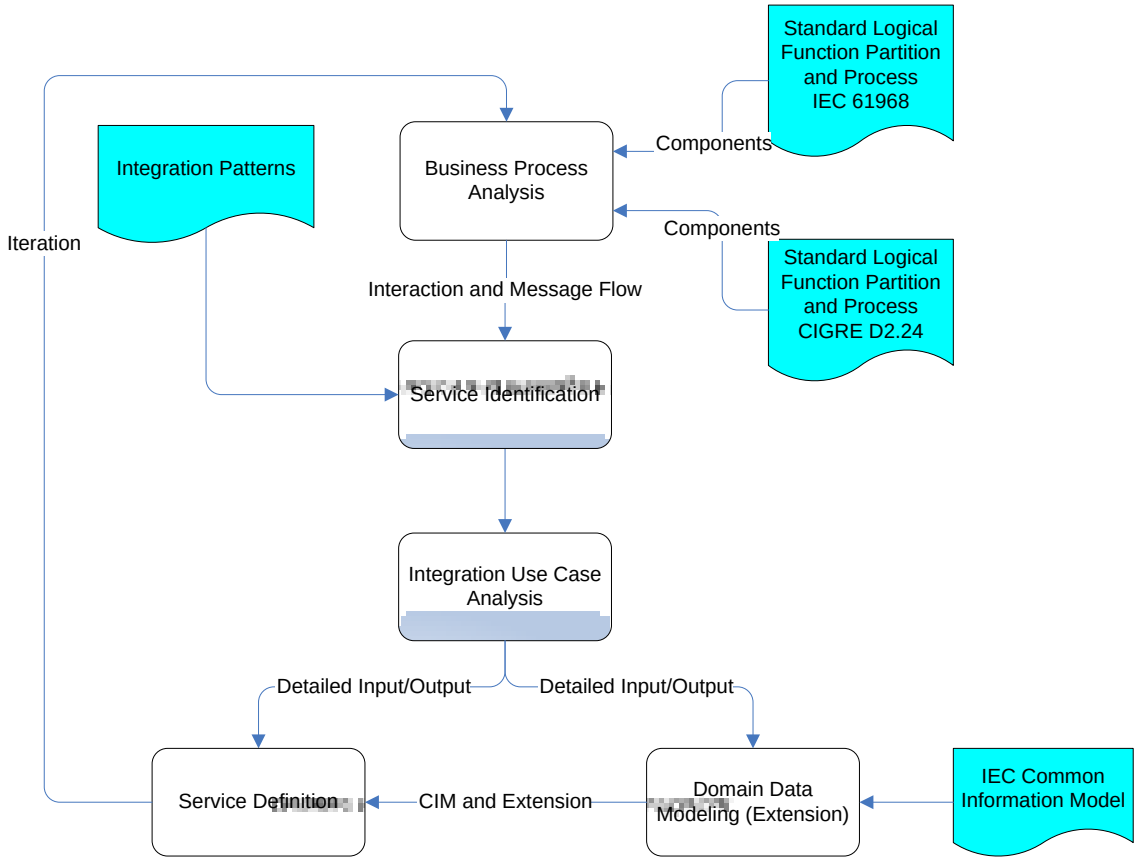

Figure 7-2: Service Design Process

The business process analysis is aimed at understanding the interactions and message flows among different components. The fundamental effort is to construct a process flow through involved components and illustrates and discovers interactions and message flow.

7.2.2.1 Service Identification

Service identification is aimed at defining a set of services with the right level of granularity, modularity, and reusability to support not only the known business processes, but also potential new business requirements and processes. The ideal case is to create new business logic and functions fully based on existing services.

The service identification starts with decoupling interactions based on Business Processes. The purpose of this step is to understand exactly what Data Sets are exchanged among different components. This step helps to identify reusable and loosely-coupled Services.

7.2.2.2 Use Case Analysis

The purpose of use case analysis is to understand requirements for each identified Service. The data requirement for each Service needs to be clearly captured and defined. The attribute level details are captured for each Data Set. If a Service has any dependencies with other services, it should be analyzed and documented at this point. Service level agreements are defined during

this phase, such as message exchange frequency, maximum allowed latency, service availability, expected response time, expected time to exchange, and security requirements.

Based on the use case analysis, necessary CDM extensions may be required.

7.3 Architecture Description

This section describes what elements the architecture is comprised of how those elements interact functionally and, most particularly, what capabilities are delivered to system implementers. All of this description is strictly functional, i.e. implementation independent. It is assumed that the sort of detail that would allow implementation of rigorously conforming software will be contained in other, more specific documents.

7.3.1 Key Information Architecture Requirements

This section introduces the key functionality requirements addressed by the Information Architecture.

7.3.1.1 Information Exchange via Standard Interfaces

The Information Architecture should support arms-length, open exchange of Data Sets between producer and consumer components.

When information flows from a producing Business Function to a consuming Business Function, our architectural objective is to minimize the interdependence of producers and consumers so that they may be designed and developed as independently as possible. This is a critical characteristic of an architecture that can grow and adapt to changing technical and business conditions.

A Canonical Data Model should be used to define formal agreement as to the structure of the information flowing between Business Functions.

Consumption of data should not require an unpredictable or excessive resource burden on the producer or the producer host environment. (This is particularly important for real-time producers.)

7.3.1.2 "Data of Record"

The Information Architecture is a comprehensive system for managing "Data of Record" across the portion of an enterprise to which the architecture is applied.

Some data is strictly a local issue within a Business Function. It has only transient value and is not exposed beyond the Business Function and the designers of the Business Function are free to handle such data in any manner they wish. The Information Architecture is not concerned with such data.

Other data has broader visibility. Its design and management require taking into account the world outside a single Business Function. The Information Architecture specifies the treatment

of such data, and refers to it collectively as “Data of Record”. Data may attain Data of Record status because of any of the following:

- The data is produced by one Business Function and consumed by others.
- The data is required to be available for ad-hoc query outside the producer’s environment.
- The data is required to be available for user interface presentations created by parties other than the Business Function creating the data.
- The data is required to be archived and/or to be available for audit of operations.

7.3.1.3 Data Mastership

Data of Record should be modularized into Data Sets, each instance of which has a single Producer.

Managing enterprise data is greatly simplified if there is clear mastership for data. The principle of modularizing data so that each module has one master is basic to the Information Architecture.

7.3.1.4 Enterprise Life-Cycle Cost

Lower life cycle enterprise IT costs will result when Data Set requirements are implemented in a consistent manner across the enterprise. However, achieving consistency adds requirements to initial implementation and could drive up initial implementation cost.

The implementation of the Information Architecture should include code to implement the required consistent treatment of Data Sets in a form that minimizes the effort for business programmers to create conforming Business Functions.

7.3.1.5 Data Partitioning within an Overall Information Model

The Information Architecture should support partitioning of the overall data model into sets of data that are produced atomically.

EMS/MMS Systems have always had challenging performance requirements. The exchange of data between state estimator and contingency analysis, for example, should not require more than a few seconds, and the fewer the better. The exchange of data between state estimator and a UI component needs to occur in under a second. And the historical system must be able to record all the solutions (hundreds per day currently) – which introduces a concern about the sheer quantity of data. These operations are potentially far more efficient if the design can take advantage of the fact that the “master model data” (describing the physical devices in the system) and the bus configuration (which results from topology analysis) usually do not change – and a new solution only needs to record them if they change. To take advantage of this, however, requires the ability to partition data into sets and to know, for example, that solution set #43103 is based on topology set #608 and master model set #71.

7.3.1.6 Business Contexts

The Information Architecture should support business contexts.

EMS/MMS systems need to support a variety of business processes. These processes are conducted in a variety of different "contexts". For example, a real-time network state, a day-ahead market, a study power flow at some future time, a reconstruction of a past event, a training scenario, a test bed, etc. Many of these processes utilize the same business services and thus produce the same kinds of data sets (although different *instances* of those sets). For various reasons, such as access by the UI and access to the historical record, it is important to know the context that produced each instance of a data set which should be part of the Information Architecture.

7.3.1.7 Events and Incremental Update

The Information Architecture should support incremental change and event notification.

For some data sets, such as SCADA status measurements or the master model, it is efficient to implement a change by updating individual items within the set. Similarly, the history of a data set may form a more compact and useful historical record if it stores updates, rather than recording the entire data set each time anything changes. As we look into the future, we anticipate increased use of event driven architecture, in which "events" (selected kinds of data changes) are available to be used to trigger processing activity. These situations illustrate that incremental change functionality is an important aspect of the Information Architecture.

7.3.1.8 Audit Trail and Historical Archiving

The information architecture should support versioning and archiving of public data.

It is increasingly important that EMS/MMS systems leave a complete, auditable trail of activity. This historical record should support accurate reconstruction of the state of the system and of all actions taken on the part of users. The historical archive itself is considered an application, rather than a part of the information architecture, but the interfaces for both saving data sets and retrieving saved data sets are within the scope.

7.3.2 The Role of the Canonical Data Model

The information architecture pursues the goal of semantic consistency by deriving the structure of public data from an overarching canonical data model, which is referred to herein as "*CDM*".

The information architecture furthermore takes a particular view about how this information model is constructed.

- The CDM is an information model that limits itself to describing the state of the real world at one point in time, not its progress through time.

- Time is covered in the information architecture by stipulating how CDM state views are versioned through time and how the changes from one state to the next are expressed as incremental changes to a state.

This is a state-oriented approach primarily concerned with analysis and control of a state. However, the architecture should satisfy the full spectrum of needs, which at a high level breaks down as follows:

- The state of the system at any point in time is presented in state-oriented Data Sets.
- The progress of an aspect of state through time – and notifications of changes – is accessible through incremental updates to Data Sets.
- Reconstruction of past states is supported by archived Data Sets in either full or incremental form.
- Historical analysis associated with an object over time – or general historical query – is by event archives.
- Hypothetical or planned future states are supported, typically using business contexts other than real-time.

In all of the above, Data Sets utilize CDM semantics.

7.3.3 Data Sets

A Data Set is a set of related data that is normally produced and maintained by one business service within a system, for general use by other services. Its content is defined by a subset of the *CDM* for the system.

7.3.3.1 Data Set Standardization

All parts of any CDM governing a Data Set should be publicly available to any potential author of a business component.

The data services described in this section are completely independent of the choice of CDM.

However, the intent of the IEC TC57 and CIGRE D2.24 effort is that the CDM will be standardized to the greatest extent practical. An Abstract Component cannot be built and installed in standard form (i.e. with lowest possible cost) unless all the Data Sets it produces or consumes have been standardized. The choice of CDM standard is the IEC TC57 Common Information Model (CIM), as referenced in Section 7.4.1.

The CDM governing Data Sets will be a version of the standard CIM, plus one or more extensions. Extensions are required wherever there are new business requirements to be met, or in any area where the standardization process has not been completed. In the latter case, extensions may actually be proposed additions to the standard that have simply not been finalized. Depending on the degree that non-standard parts of the CDM are used in any particular Data Set, the amount of work to make a using business component interoperate can be expected to be correspondingly larger. Note that *proprietary* extensions of CIM, where the owner places restrictions or conditions on the use of the extension, work against the goals of the Information Architecture and are discouraged.

7.3.3.2 Data Set Specification

A Data Set specification is a subset of the CDM, and therefore is an information model defined in metadata. A Data Set specification determines the content of Data Set instances which obey that specification.

A Data Set specification includes:

- Data Set specification identifier.
- Data Set specification version.
- Underlying CDM version designator. In practice, the Data Sets in use in a particular installation may not all be derived from the same CDM version.
- Included CDM classes, including:

  Specification as to whether the Data Set allows producers to add or delete instances of each included class.

  Specification as to which CDM object attributes and relationships of each class are included.

- A specification of dependencies on other Data Set specifications – i.e. situations where one Data Set instance can only be created in relation to an existing instance of a set that it depends on.
- *A Data Set instance may contain attributes of objects that are created in other Data Set instances. (In other words, attributes of a class can be "split" among different Data Set specifications.)*
- *A Data Set instance may contain new objects that have relationships to objects created in other Data Set instances. (In other words, relations of a class can be "split" among different Data Set specifications, as well.)*
- *Restriction: A Data Set (both instance and specification) dependencies are directional and not circular.*
- *Restriction: At least one Data Set (both instance and specification) in a system is a "root" which has no dependencies. (This is typical of "master model data" that is used to configure a system.)*

7.3.3.3 Data Set Instances

An instance of a Data Set, in addition to conforming to a Data Set specification, is also distinguished by a business context and a version.

7.3.3.3.1 Contexts

The business "context" normally corresponds to the purpose of that instance, and is normally established by a business process orchestration that directs activity. Essentially, every service

is conducted within a business context that defines how its Data Set instances are distinguished from those obeying the same Data Set specification but produced by other business processes.(as in real-time versus test versus study contexts.) The business process orchestration usually sets the context.

In terms of information functionality, however, context simply means that Data Set instances are distinguished by a context name.

7.3.3.3.2 Versions

Any contextual instance of a Data Set may furthermore have versions which describe the instance as it changes over time. New versions may be created incrementally – by issuing only updates to the previous version. Versions are identified by a primary sequence number indicating the *full* Data Set version and an incremental sequence number indicating the incremental updates. Versions commonly represent a specific point in time.

7.3.3.4 Object Identification in Data Sets

In IEC standard CIM, objects inherit a Master Resource Identifier (MRID). This is meant to be a unique tag used by software as the formal identifier. MRIDs are opaque and not meaningful to humans.

Objects are always identified by an MRID that is unique across all objects in any instance of a Data Set.

When two objects in different instances of Data Sets are representing the same real world thing, they should carry the same MRID.

All data access interfaces in this architecture use MRIDs as the data binding to individual object instances, rather than names or pathnames.

MRIDs designate the thing being represented, not the instance of the representation. Thus an MRID is not a globally unique software object identifier – it is a unique real world thing identifier within the scope of business processes that have agreed to share identifiers. We make the constraint here, however, because of the way that Data Sets are used, that within any Data Set instance, MRIDs will be unique.

The creator of an object assigns the MRID. In IEC standard CIM, these creators are called Model Authorities and all objects are related to a Model Authority. In the case of a dynamically created object such as TopologicalNode the Model Authority may be a software entity, but in the case of master model objects, a modeling software entity is representing some user who is assigned as Model Authority.

An important use case is the import of objects because it raises the following question: if a set of objects is imported into a system, is this creation? The answer is that when an object is imported, it should normally retain the MRID that was assigned, which makes import different from creation. The importation use case means that MRIDs used in a system may have come from external sources. This implies that there be an agreement that guarantees that MRIDs generated in one system will not duplicate those generated in a cooperating system. Model Authorities are designed as the vehicle for making such agreements within power system models.

7.3.3.4.1 Names

Names are regarded as data. Names can change like any other data – and should not be relied on as identifiers. MRIDs are used by software for identification, but have no significance to human beings.

Names (and sometimes other data attributes) are, however, obviously what is used by human beings to distinguish one object from another. Therefore, in any process in which a human being creates a reference to an object (such as linking a UI data field to a data item) the human is guided by the name of the object and its related objects (often the names of its parent objects in a hierarchy). Any sequence of attributes that locates an object is called a “path name”.

Path names may sometimes be retained in circumstances where references are being created between data sets such that the referential integrity of the relationship expressed as an MRID is not always guaranteed. One example is the linkage of a UI data field to an element in a Data Set. Another is a reference from an object in an imported Model Authority to an object in a Model Authority from a different source. Retention of path names allows linkage problems to be diagnosed more easily. They also allow linkages to be re-built if the MRIDs have not been synchronized.

7.3.3.4.2 Referential Integrity

Referential integrity in a database refers to maintaining references between objects as elements in the database are created, deleted or modified.

Referential integrity in this information architecture is approached as a broader issue:

- Data Sets may have references to other Data Sets, not just internal.
- References (like data bindings from the UI) may exist that are not represented at all in the CDM.
- Path names may be stored in addition to MRIDs (as an aid to debugging or reconstructing references).
- Different references used by applications may be correlated within a component such as a naming service.

MRIDs are the universal basis for maintaining referential integrity in this information architecture, but it is important to recognize that MRID references are not perfect. It is, for example, not possible to guarantee that two objects with two different MRIDs are not created for the same real world entity. If this kind of error gets into operations, and is subsequently detected, a simple deletion of one of the references would probably delete some legitimate historical data and/or break some path name references.

The facts that references between objects are sometimes set erroneously and that names can change over time, invalidating path names, assure that fully automated maintenance of perfect referential integrity is not possible. The MRID-based system presented here is therefore presented as a good solution, but one for which all parties should recognize the limitations and supply tools for diagnosing and repairing the occasional problems.

7.3.4 Information Flow

Data Sets are the vehicles that carry information from producing sources to consumers that need the information. The Information Architecture specifies the nature of this information flow.

The reference architecture presumes a single writer (the producer) per context for a Data Set. A single Producer therefore has data mastership over each series of Data Set Versions produced in a context.

The above captures the idea of data mastership – Data Sets originate with a single Producer. However, in a distributed computing environment, different Data Sets may be produced by different Producers running in different hosts, and Consumers may be located in any host in the environment. Our chief objective in managing this information flow is that the details of information flow are transparent to the authors of the Producer and Consumer Business Functions. Producer authors just concern themselves with producing the Data Sets. Consumer authors just concern themselves with asking for Data Sets. In general, implementation of common requirements for treatment of Data of Record (including backup, archiving, etc.) are removed from the responsibility of Business Function authors and handled at a system level.

What might be called the "mainstream" implementation of information flow for this Information Architecture is an environment in which hosts are connected to a message bus infrastructure and where Data Sets become message payloads. However, many situations will call for variations on this model and Data Sets will also be moved by other means. For this reason, production and consumption of Data Sets must be cleanly separated from other Data Set handling.

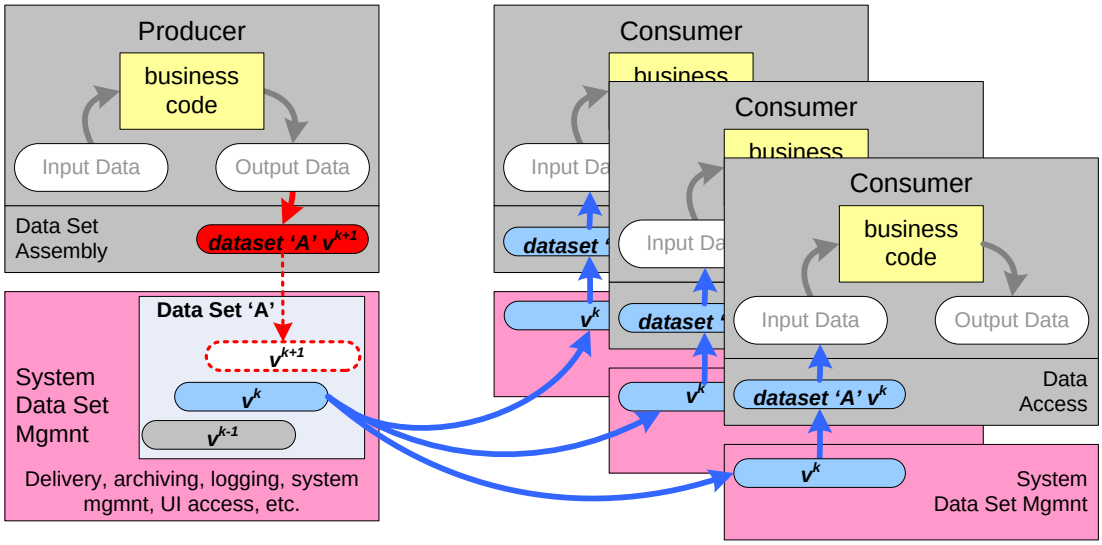

Figure 7-3: Basic Producer / Consumer Pattern

The Information Architecture is based around the ideas shown in Figure 7-3.

For Producers:

- The content of a new Data Set version is created by the business logic of the Producer Business Function. Initially, this data is in whatever form is effective for the business logic.

- Output data is made available to the outside world by creating a new version of a Data Set.

- When the Data Set is complete, it is closed and the new version comes under the control of the system Data Set management, which makes it available to Consumers.

For Consumers:

- Consumers make requests to system Data Set management for access to a Data Set.

- The system responds by establishing connections to the requested Data Set, usually the latest completed version.

- The connections allow the Consumers to gain access to the content of the Data Set.

The connection between Producers and Consumers is managed generically by system data management function, rather than by specific Business Function logic. In addition to Producer-Consumer connections, this system data management also handles other important Data Set requirements, such as backup, archiving, logging, and UI access.

7.3.4.1 Performance Requirements and Data Exchange

Data Sets act as interface vehicles in an SOA environment where the primary goals being served are all life-cycle cost oriented – goals such as, simplifying application code, increasing the independence of one application code from another, facilitating the integration of applications from different vendors, facilitating upgrade of components, etc. In power systems operations environments, however, there are also crucial performance and efficiency requirements. Despite continuing gains in computing resources, we are not yet getting close to the point where these concerns have disappeared. Instead, there is an intense ongoing competition for every increase in resources.

Unfortunately, SOA and standard interfaces and well-structured code are one of the main competitors for resources. It is usually true that gains in the life-cycle development costs come at the expense of layers of structure that consume more resources than their “harder-coded” counterpart designs. This is certainly going to be true for a Data Service that supports practical standardized multi-vendor application interaction.

For most application-to-application exchanges, the tradeoff of increased time to produce and consume data for better structure and more efficient development is a good tradeoff. For a few situations, it will seriously impact the delivery of critical information to users in a timely manner and it will not be a good tradeoff.

The Information Architecture is intended to be rigorously applied only where performance is maintained at an acceptable level.

Where performance requirements are critical, more efficient means of exchanging data are required. In these cases, the Information Architecture should be taken as a vision that is to be achieved to the degree that is practical. For example, a particular performance-oriented consumer might tie itself directly to a producer (in terms of design) in order to use a high-speed form of exchange, but the producer could also go on to produce the same information in standard form at lower speeds for other consumers. Implementations that allow such high-performance exchange are encouraged, as long as the faster interface is also open, and the standardized interface is also supported.

7.3.5 Producing Data Sets

This section describes the Data Service interface used by the producer to create Data Sets.

7.3.5.1 Concepts

Data Sets divide the CDM into sub-models that are created by a given producer instance. (Thus a modeling application produces the master data sub-model, a SCADA application produces a measurement sample sub-model, a topology application produces a topological bus sub-model, and a state estimator application produces a steady-state solution sub-model.)

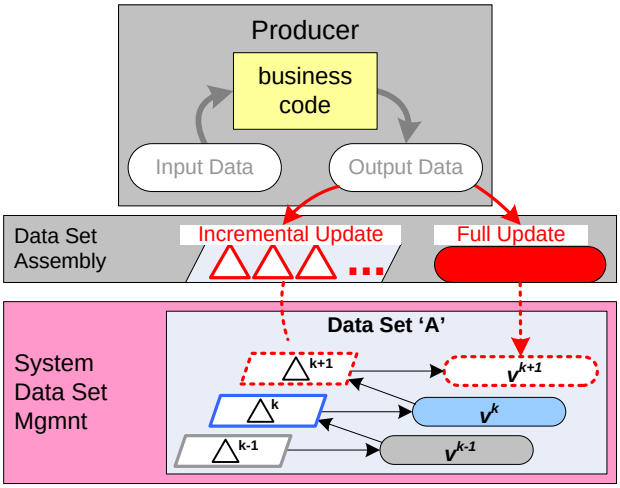

Figure 7-4: Producer Functionality

Figure 7-4 shows greater detail about the Producer's functional interaction with Data Sets. What a Producer always does is to package a "transaction", which is either a set of changes from one version of a Data Set to the next (an incremental update) or a complete new Data Set (a full update). Incrementals are specified using add, modify, delete operations. In all cases, data is not released for consumption until the new version is closed by the Producer. After closure, the producer's work is finished and all responsibility for persisting and distributing the new Data Set version belongs to the Data Set management.

The Producer encloses all changes within transactions, with each transaction producing a new version of the Data Set.

This approach does not preclude shared writing – it simply provides no support for it and leaves to shared writers the responsibility of coordinating so that it looks to the rest of the world as if there is a single writer.

If a Producer is interrupted having provided some data, but having failed to publish the transaction, none of the data should reach any consumers.

System Data Set management should be able to maintain two ways of "seeing" a Data Set:

- The incremental view gives the set of changes from one contextual version of the Data Set to the next.
- The full view shows the complete Data Set instance.

The producer of a Data Set is responsible for:

- Identifying the data context that it is modifying.
- Providing the new data content.
- Publishing the data at the time when a new Data Set is complete and usable.

7.3.5.2 Data Set Management Functionality required by Producer

7.3.5.2.1 Specifying a Data Set Instance

The Data Service should support declarations that provide a producer with (read and) exclusive write access to a specific context's Data Set and prevents conflicting producers.

7.3.5.2.2 Specifying Transactions

The producer for a Data Set controls its transactional nature.

The Data Service should support "full/incremental" mode that declares whether this transaction will completely specify the next version.

The Data Service should support a publish operation that marks the end of a transaction and signals that the resulting state is a new version ready for consumption.

Each transaction closure declares the result as a new version of the Data Set.

7.3.5.2.3 Data Set Update by the Producer

Whether a new version is fully re-created, or whether it is created by updating the previous version, the producer uses incremental operations to build a transaction. Incremental operations may be formed into named "events". Incremental views of a Data Set update typically consist of multiple events. For many Data Sets, it is important to create a historical record of events for potential consumers, and so the structure used in modifying the data is important. Example: A

topology processor produces a record of equipment outages as it produces the next system topological state. It is more convenient to look through the incremental history for outage events than it is to compare topological states.

7.3.6 Consuming Data Sets

This section describes the Data Service interface used by consumers for receiving Data Set information.

The reference architecture presumes that consumers do not see any results until the transaction is closed, and multiple consumers per version are allowed.

This does not preclude a more intimate connection that shares data without formally creating a transaction, as might be employed between a SCADA update and certain processors of SCADA. It merely means that there is no support in the architecture and such users should coordinate among themselves as necessary. Usually, this would mean that they will periodically need to produce a new version that looks like a transaction from a single source.

7.3.6.1 Concepts

Data Sets divide the CDM into sub-models that are created by a given producer instance. Contexts generally consist of a related set of Data Sets, the union of which is still a logical subset of the overall CDM. When *any* of the contributing Data Sets is updated to a new version, the new *net* result is considered a new version of the context. If the context tracks time, the new version represents a new point in time. The union of the Data Sets available within a context defines a "context state".

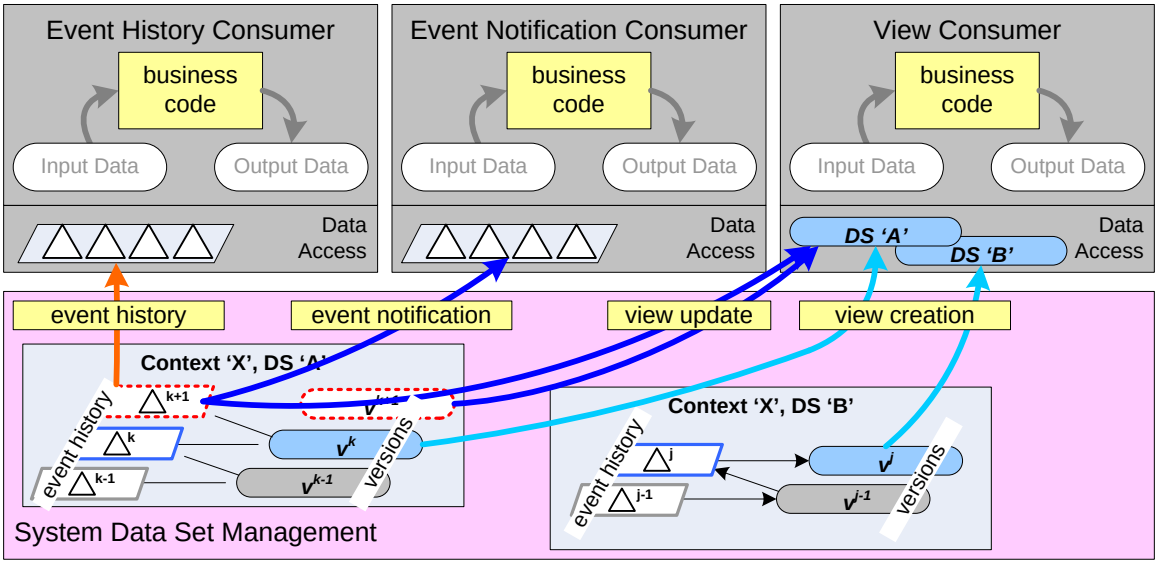

Figure 7-5: Consumer Data Set Access

There are several important patterns of interaction between consumers and the Data Service, which are illustrated in Figure 7-5 and discussed in the following subsections.

7.3.6.2 Data Service Functionality required by Consumer

7.3.6.2.1 Specifying a Data Set Instance

The Data Service should support an "Open as Consumer" operation that provides a consumer with read access to a specific Data View.

Normally, access is granted to the latest version of the context state that has been completed by the producers. Read access is shared among all consumers by default.

7.3.6.2.2 Defining a Query

It is normal in this Information Architecture, that data is acquired through a "standing query" – in other words, that an access request is defined and then refreshed repeatedly as the data changes. For this reason, definition of the query, which occurs once, is separated from delivery of data.

Queries identify data either by fixed specifications, or by some browsing interface that allows exploration to locate desired data. Some specifications involve only the generic class-attribute-relationship metadata; other specifications locate instances by MRID or by path names or by any exploration that ultimately provides a "that one" designation from a user.

The functional result of defining a query is a mutually agreed upon locator that is understood by the Data Services and the data consumer.

7.3.6.2.3 Delivering Data

The data service should support consumption in either incremental or query form, regardless of how the data was produced.

Ultimately, the business code in a consumer has a choice of two ways to receive data.

- Incremental delivery supplies information as a set of changes, expressed with embedded CDM metadata (such as XML).

- Query delivery supplies information as a response to a query, where the query contains the CDM metadata.

Basically, either form is capable of accessing any of the information in a Data Set. Individual applications may find one or the other to be preferable in specific situations.

7.3.7 Information Interface Profiles

An information interface profile is a specification that governs a functional interface. There may be hundreds of services and hundreds of Data Sets involved in creating the set of systems that are under the governance of a CDM. It would be very awkward if a change in the CDM necessitated review and revision of every interaction with every Data Set. Profiles (which have been mentioned earlier in this document) are the key to practical evolution.

In a given system implementation, all Data Sets are derived from CDM, but not all Data Sets are derived from the same version of the CDM.

The idea is to divide the set of all interfaces into profiles, each of which governs one or a few Data Sets that have closely related functionality and (usually) a smaller community of users (producers and consumers). Implementation is governed by the profile. Profiles need not change simply because CDM changes – change in a profile is governed more by the needs of its community of users.

Every archived Data Set (full or incremental) should be labeled with the version of the information interface profile under which it was produced.

The heart of the profile is the information model for its Data Sets, but it may also include other constraints and restrictions – generally provided that the overall result is still manageable within the Data Services.

7.3.8 Managing Data Sets

Additional Data Services are provided for managing Data Sets so that these services do not have to be implemented by producers or consumers.

7.3.8.1 Concepts

In this architecture, producers and consumers are not directly connected. Indeed, they have no explicit knowledge of one another. Instead, producers make data available via a Data Service using a standard interface, and consumers get data as clients of the data service using a standard interface. The clients do not need to know who produced the data, or how it is managed – they only need to know what data they require, which is specified as version, Data Set and business context (and where version commonly defaults to the latest version and business context commonly defaults to the context in which the consumer was invoked) and how to find the relevant data service to use. Architecturally, as much data management-related functionality as possible is put into the Data Services rather than the producing or consuming business services.

The Data Service manages Data Set instances that are defined by profiles of the CDM metadata. Since this is essentially persistent data defined by metadata, it stands to reason that an implementation of Data Services might utilize some form of database, connected to producers and consumers by some form of distributed access. The architecture, however, is not concerned with this implementation – it is only concerned with the services that are supported.

7.3.8.2 Data Service Functionality

The Data Service should provide the functionality specified in the following subsections.

7.3.8.2.1 Metadata

The Data Service should provide tools and methods for maintaining the CDM and for deriving Data Set specifications from the CDM.

7.3.8.2.2 Persistence

Data service should make it possible to persist all versions of a Data Set for an arbitrary period of time.

No statement is made about what form such an archive will take, or whether the most recent and active instances are stored in the same form as older data. Services dealing with archives are expected but for the most part, left open to competition. Examples of such services would be flexibility in declaring how many versions are automatically retained, ability to manually save versions that are not automatically saved, etc.

7.3.8.2.3 Contexts

The Data Service should provide a way to instantiate a context and to initialize the content of a context's Data Set instances from any saved version of Data Set instances.

Contexts are associated with a business process and the state of a context is maintained in Data Set instances. For each context, there will be one or more Data Set specifications, each versioned over time.

The capability to initialize from archives is critical to reconstructing past states for further analysis, and for studying hypothetical states.

7.3.8.2.4 Access Control

The Data Service should grant access to Data Set instances by producers and consumers based on permissions associated with the invoker of the producer/consumer. The Data Service should conform to the provisions of the Security Architecture in granting such access.

7.3.8.2.5 Backup

The Data Service should make it possible to create backup copies of Data Set versions as they are completed.

7.3.8.2.6 Data Set Events

The Data Service should support notification to consumers of the completion of a transaction by a producer.

A closure event may optionally contain either full or incremental XML describing the new version of the Data Set instance. It also may optionally use a "claim check" pattern of delivery (designed to keep messages short) where full or incremental XML files are referenced, but not included in the message.

The Data Service should allow consumers to subscribe to any selected set of add/modify/delete operations on any CDM class in any context.

The message bodies of update messages are formatted in incremental XML.

7.3.8.2.7 Data Set Sampling

The Data Service should allow specific data items within a Data Set to be sampled on a periodic basis.

The sampling service should be able to collect samples in a holding area and supply to users either via messages on an enterprise bus or for direct access (perhaps via a claim check pattern).

Data sampling is designed to support functions like trending or data historians.

7.3.9 History and Time

A context in this architecture is used to collect together Data Sets that represent the state of a business process for one instance of that process. Although there are situations where a business process has no sense of time passing, and may be non-specific as to what point in time is being represented, it is nevertheless important to think of contexts as the architectural vehicle for tracking a business process as it progresses through its states. As things change, the changes result in new versions of one or more Data Sets within the context – and thus a new net version of the context state. Context activity is based on the latest versions of all the Data Sets, and a complete historical record is achieved if all versions of the Data Sets are archived for as long as they are needed.

Naturally, it is important to record the actual time at which changes of state take place. There are, however, important business processes that are simulating a different time, and that need their own context clock keeping context time. Consider, for example, the need to reconstruct a historical situation for post mortem analysis. An analytical simulation should show a time in the past and its "clock" may be slowed down or sped up from its real-time pace to suit the analyst's need. Clearly, a context clock is important here and it does not correspond to a real world clock.

Context time should be thought of as the progression of states, rather than literal clock time. It will be normal that there is a "real-time" context that is trying to track the real world, but even here, if we were to be very precise about it, there is a delay between the real world time and computer simulation of the real world. "Context time" is the simulation time. (Within a real-time context, one might find time-tagged measurements that have skew and do not correspond exactly with the context time – but the time tags are "real world clock samples associated with the measurements being used for simulation of the current context time".)

Business process usually focuses on the latest context state, but past context states from one business process are often used to initialize or reset another business process – as when a simulation context is initialized from a past state of the real-time context.

Saving consecutive versions of Data Set instances is a straightforward way to preserve a complete record of activity, but is it practical? This question can be asked from two standpoints – does it produce too much information? And, is the stored information usable?

For use cases that require reconstruction of past states, being able to retrieve a saved context state is a practical answer provided the storage is available and there is enough bandwidth – and these provisos can be set aside simply by making the archiving selective and dynamically controllable.

There are, however, other common use cases for historical data in which the saving of copies of Data Sets is inadequate because it is too difficult to use. Imagine, for example, the need to find out how many times a given circuit breaker operated over the last 10 years. This is impossibly awkward to answer if one has to plow through saved Data Sets. It is more reasonable to answer, though, if one could query a continuous history of updates to the SCADA status

telemetry – such as could be easily created if the incremental updates of that Data Set are recorded. Such a record is not inordinately large, since breakers don't operate that often.

Analog telemetry presents a more specialized problem. Here, we want to serve queries of the type that ask for history on a given measurement over a long period of time, but the number of individual updates (because analogs change all the time) is enormous and needs both special processing bandwidth to record and special storage techniques to store and support retrieval. Specialized historian products have been built based around this particular requirement. These products also will store less frequently changing information, but they justify themselves on the measurement recording use case.

Thus we arrive at the following conclusions:

The Data Service should provide for saving Data Set instances from active contexts into archives (saved cases) and for retrieving saved Data Set instances into active contexts.

The Data Service should provide a historical data recording function which has the bandwidth to keep up with large amounts of real-time data. A corresponding retrieval function should allow access to any analog through time, or any set of data at one point in time.

The Data Service should provide for saving Data Set events from active contexts into archives and for retrieving Data Set Events.

7.4 Applicable Standards

CIGRE D2.24 reference architecture aims to provide utilities with uniform specification language that will allow the utilities to implement conforming systems – or, to put it in more realistic terms, to get as close to the architectural vision as is practical and cost effective, given the offerings available at the time. This section offers specific guidance as to on the use of standards that exist or are in development.

IEC Working Groups 13, 14, 16 and 19 from IEC TC57 have been working on Information Architecture issues for some time. These groups have produced two main categories of work:

- The IEC Common Information Model (CIM), which provides a single unifying semantic model for power system operations expressed in UML.

- Various sets of APIs and document-based data exchange mechanisms that use the CIM to provide implementers what they need to achieve interoperability.

7.4.1 IEC Standard CIM as CDM

The IEC standard CIM should be used wherever it satisfies the business requirements for a particular Data Set specification.

The IEC TC57 working groups have published a standard common information model, CIM, for the power industry that is consistent with the goals of this Information Architecture. This is a

"standard", but it changes continuously as new projects reveal new requirements or greater detail about existing requirements. The current plan is for annual releases in a UML format.

A key goal of the CIGRE D2.24 architecture and the CIM standards community is that business components from different developers can be expected to interact with one another – ideally without modification, but certainly with minimal and non-intrusive modification. This goal can be achieved completely, for a given Data Set, when the following conditions are met:

- The Data Set specification includes only data defined in a version of the IEC standard CIM.

- The interoperating business components have been tested against this same CIM version.

IEC standard procedures for extending CIM should be used whenever extensions to the IEC standard CIM are required.

Business requirements are always on the move, however, and many systems will of necessity venture into territory where standards do not exist. Waiting for the standards to be extended is typically not desirable. Extensions to the IEC standard CIM therefore should be viewed as normal occurrences and implemented following the procedures defined by the IEC.

CIM changes should be submitted to the IEC as candidate standards wherever there is general value.

Modifications to the IEC standard CIM should be judged for value in improving the standard. If there is general value, the change is implemented locally first, but with the idea that it will be submitted to the standards committees, and with the further idea that when it is finalized by those committees, the host system will be updated to the final modeling solution. This practice will reduce long-term maintenance costs by keeping implementations close to standards. Modifications with strictly local value are to be avoided or carefully modularized, since they will require ongoing custom maintenance as the system and standards evolve.

Extensions to CIM are preferred over modifications of the existing structures only if extensions are equally effective in addressing the long-term goal of semantic quality.

The Data Services supporting this Information Architecture should be designed to accommodate extensions and modifications of the standard CIM.

In an ideal world, where all CIM modeling is done with perfect insight, all changes to the CIM would take the form of extensions (i.e. additions only). Extensions have the happy quality that they do not force recoding of existing software. However, in the real world, modeling is an imperfect process and exclusive use of extensions will typically destroy the semantic quality of a model fairly quickly. To avoid this, CIM modifications will often be required, even though this tends to raise the initial cost of implementing the change – a very important principle that is often difficult to enforce.

7.4.2 Interoperable Business Components

The ability for vendors to construct business application products based on standardized interfaces has long been a vision of standards work – it is also a very challenging objective that can be discussed in two distinct parts:

- "Data interoperability" is is addressed by this Information Architecture, and is achieved when components produce and consume only standardized Data Sets via standard Data Services.

- "Integration interoperability" integrates a component within its system infrastructure, dealing with issues like message buses, error logging, high availability schemes, business process invocation, etc.

Of these two parts, data interoperability is the one that is more deeply entangled with the design of an application. Generally speaking, if data interoperability is achieved, but not integration interoperability, "cost effective integration" may still be achieved.

7.4.2.1 Data Interoperability

Requirements for data interoperability in a given business data exchange are specified in an Interoperability Profile. The Information Architecture identifies two basic ways that an interface can be defined as the basis for interoperability:

- **Data Set Interfaces.** A Data Set or a Data Set incremental update can be transported to a Consumer in some standard format (such as CIM/XML) by some transportation method (such as a message bus).

- **Data Access Interfaces.** A Consumer may use a stipulated set of standard data access (data query) services to obtain data from Data Sets.

The choice of interface may be different with different profiles. The IEC working groups have defined standard means for each of these, but no limitation is imposed that would restrict a specific profile from using any means that is appropriate.

Each Interoperability Profile should state the interface mechanism required.

In effect, this important requirement says that there is no requirement for information interfaces unless specified as part of a specific Interoperability Profile.

7.4.2.2 Data Set Interface – Existing Standards

For Data Set interfaces, a Consumer will take delivery of a Data Set version, or an incremental update to a Data Set version, in its entirety. How the Consumer will access the received data is not specified as part of a Data Set interface.

An example of an Interoperability Profile that uses this type of interface is IEC 61970-452 which specifies how power system network models may be exchanged.

The advantages of Data Set interfaces are:

- It is the simplest method of defining interoperability.

- If the receiver ultimately needs all or most of the data, Data Set interfaces are the most efficient method of defining interoperability.

7.4.2.3 Data Access Interface – Existing Standards

For Data Access interfaces, the Consumer's formal interface is a set of services for querying Data Sets, as described in the Consuming Data Sets section. The counterparty in interoperability is obligated to answer each query, but whether there is any transport of source Data Sets is not specified.

There are currently no specific IEC Interoperability Profiles defined for this type of interface, but there is a set of documents, IEC 61970-402, 403, 404, 405, 407, that define generic data access mechanisms that could be used.

7.5 Overview

Energy market systems and energy management systems are increasingly composed of complex functions provided by multiple vendors and multiple phased development projects. The recognized need for efficiently integrating these functions, while supporting critical, ongoing, and evolving business functions, has placed a very high priority on application integration support. The Integration Architecture is designed to facilitate these integration activities and provide for efficient, reliable, real-time operation of these systems. A major theme for the Architecture is to assume that the system will have integration requirements from the beginning, rather than adding them on as an afterthought.

The Architecture is based on Service-Oriented Architecture (SOA) concepts**.** The Architecture intends to leverage the considerable IT industry investment and rapidly advancing capabilities within the SOA domain. The Architecture describes how functions are to participate in the system using SOA to its advantage while meeting practical control room requirements such as high performance and availability.

7.6 Scope

The Integration Architecture addresses the following topics:

- **Functional Partitioning** addresses terminology and organization of major systems, and their component functions. Examples of functional components are given, but these examples are not intended to be a specification of which components are part of this architecture. The concepts of service composition and components are illustrated.

- **The Interface Architecture** describes the standards and patterns of the adaptor based interfaces. The adaptor patterns may be used to interface new or legacy applications into the architecture in a compliant manner.

- **The Service Architecture** describes service specifications required for interoperability and configuration to support business processes.

- **The Integration Layer** contains the description and function of the central integration framework at an interface level. The focus is upon the application interfaces and the abstraction of the communication layer. Specific functions of the integration tier are also discussed.

- **Information Exchange Pattern:** Applications may choose or in fact require several standard messaging patterns. Examples are given of several common exchange patterns that should be supported.

- **Status, Monitoring, and Error Reporting:** Application status is to be made available for central reporting and system management. The system supports the ability to monitor its activities both from a functional and security standpoint. Any component of the system must be able to report its own health and readiness in order for the system as a whole to function properly.

The Integration Architecture does not specify specific choice of integration layer, hardware, or software platform.

7.7 Architecture Description

This section describes key concepts and design principles of the architecture and how they relate to the Architecture as a whole and provides a framework from which to create compliant interfaces specification. The work borrows heavily from the ISO/RTO Council's Technical Reference Architecture.[3]

7.7.1 Key Design Principles

Services have explicitly defined interfaces. Interfaces are formally declared and known so they can be implemented clearly and integration risk is minimized. The use of WSDL enables a clear and formal definition of the interface.

Services are loosely coupled. Services are not dependent on internal interfaces or other dependencies beyond the standard interfaces. This enables replacement or modification with reduced risk of unexpected side effects. This also enables replacement and competition among vendors of standard components. Additionally specific legacy or "best of breed" components can be integrated into a system which does not conform to the internal architecture of these components.

Services use abstraction. Services are abstracted from such issues as technology choice and quality of service. The choice or modification of such abstracted qualities is possible without

[3] The ISO/RTO Council (IRC) is an organization of North American system and market operators. Starting in 2006, the IRC embarked on a project to develop architectural standards and application interface specifications for its members. Details may be found at www.isorto.org.

change to the service components. For example, if the transaction rate of a service is required to be doubled, that service can be easily scaled up and possibly the communication infrastructure can be scaled without other modifications to the system. Such abstractions enable effective migration of a system as requirements change.

Services are reusable. Services are reusable in different business contexts and possibly across wide ranging service compositions. Clearly this is a chief objective ultimately resulting in cost savings and improved system consistency. The Information Architecture works in conjunction with the Integration Architecture to reuse service implementations within different contexts or domains. For example, services should be reused within different business processes in the on-line system, again in similar processes but within the context of a market simulation, or again within the context of a development and test system.

Services have autonomy. A service should have control of its environment (such as the hardware and its resource usage) in order to best provide the quality of service required. If services are autonomous in this regard the optimal choices for resource allocation for system benefit is more easily realized.

Services strive to be stateless, but handle state effectively when required. The Information Architecture helps to address this issue in many situations. It is well known that stateless services are unrealistic to many of the high performance services typically required, but these states can be formalized and in fact made part of the service interface by using the concepts outlined in the Information Architecture.

Services are discoverable and understandable. Services can be found at design time and at run time such that existing functions are reusable. The common information model used for the Information Architecture and the service contracts help make services understandable.

Services can be composed. Services are well behaved and can be composed and orchestrated in new and creative ways to achieve business value. Custom service composition to meet specific business needs is typically required and should be efficient to implement and maintain.

Services are managed. Services are started and initialized by a well defined interface. This is part of management of the system and provides for efficient and standard control of the system.

Services are always in a known state and that state can be requested. Here the state refers to the services ability to provide its services. This is required for monitoring the health of the system and initiating system failover, security precautions, or other system decision processes.

Services leverage an integration layer. Services do not communicate directly to other services, but rather use an integration layer to provide a potential abstraction. This is in support of several other key objectives such as abstraction, reusability, and discovery.

Services can emphasize specific interface requirements. For some services high performance is a critical requirement and possibly other objectives might be compromised to achieve this requirement.

Services are testable. Services should be testable independent of the complete system. Such testing is required for quality control in deployment of new services or new versions. This also improves development efficiency and debugging of potential implementation or design defects.

7.7.2 The Interface Architecture

The adapter interface architecture described in this section borrows from the ISO/RTO Council's Technical Reference Architecture and follows a service oriented architecture that is based upon the authoritative architectures of both the World Wide Web Consortium Web Services Architecture[4] (W3WSA) (and supporting standards in WSDL 2.0[5]) and the IEC 61968-1 architecture framework. The layering concepts and definitions contained in sections 4 and 5 of IEC61968-1 (Interface Architecture and Interface Profile respectively) should serve as authoritative source for the core adapter architecture while W3WSA and WSDL 2.0 should serve as the authoritative source of the service oriented architecture characteristics.

NOTE: Although WSDL 2.0 has been chosen as a formal description language for service specifications, there is no requirement expressed or implied that implementers will use web services technology during implementation in order to comply with these standards.

As depicted in IEC61968-1 there are two logical adapters located between a Component and the Middleware Services, the Component Adapter and the Middleware Adapter. Another layer, identified as Interface Specification, is located between the Component Adapter and Middleware Adapter. A complete diagram of this relationship, as defined by IEC 61968, is shown in Figure 7-6.

---

4 Booth, D., et al, Web Service Architecture, Wide Web Consortium, 2004-02-11; http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

5 Chinnici, R., et al, Web Services Description Language Version 2.0, World Wide Web Consortium, 2006-03-27; http://www.w3.org/TR/2006/CR-wsdl20-20060327/

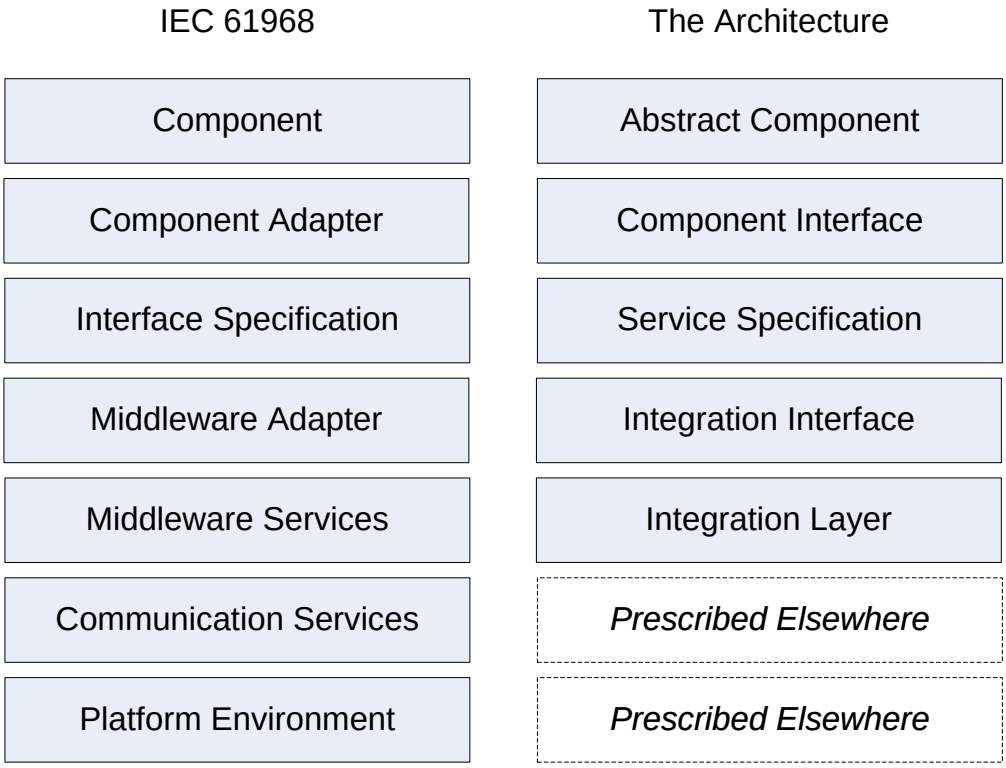

Figure 7-6: IEC 61968 "Layers" Comparison

The component Interface refers to that portion of the adapter architecture that delineates the business functions provided by an application, from the service interface operations provided by an adapter. A component interface describes the orderly interaction between an adapter and an application.

All interactions between a component and an adapter should be fully documented in human readable form (such as OpenOffice, Adobe PDF, HTML, or Microsoft Word) using a formal definition language (e.g. WSDL, DDL). Included in the documentation will be definitions for all data elements.

7.7.3 Service Specifications

The objective of a Service Specification is to facilitate interoperability among Components via Services. A Service provides functions on behalf of an Abstract Component and is manifested as a software program or programs (referred to as "agents" within W3CWSA) that work in conjunction with one or more business Functions.

Each Service Specification should be:

- Declarative with regard to business data elements, information exchange model, status/error information, security (referred to as semantics, operational behavior, status conditions, elements, attributes, operations and operating parameters in IEC61968-1) for all the services that are part of a service interface specification and where appropriate comply with the Information Architecture

- Programming-language neutral
- Independent of any specific middleware technology
- Compliant with the naming conventions prescribed in the appendix “Naming Conventions”

Any message definition or service description may be extended by an implementer provided that any such extension is properly identified using an XML Namespace that is different from that used by this architecture.

Services should be capable of readily being composed into business processes with the aid of standard tools from Integration Layer product vendors.

7.7.4 Integration Interfaces

The Integration Interface describes the abstract functional characteristics that enable services to perform inter-component interactions via an integration layer. The integration interface makes it possible for a service to send and receive information through an integration layer, regardless of the specific technologies employed within the integration layer.

Integration interfaces should support the commonly available integration layer technologies. Possible examples include:

- Message brokers
- Message Oriented Middleware (MOM)
- Hypertext Transfer Protocol (HTTP/HTTPS)
- Message-Queuing Middleware (MQM)
- Enterprise Service Bus (ESB)
- Business Process Management (BPM) Servers
- Relational Databases
- Object Request Brokers (ORBs).

The Architecture makes no recommendation on which of these should or should not be used,

7.7.4.1 Integration Layer

The Architecture makes several assumptions regarding features of the integration layer.

The integration layer should support:

- Reliable communications
- Defined business service level agreements

- Defined transaction volumes
- Message transport characteristics (defined in the Service Specifications)
- Message Exchange Patterns (MEPs) (*see "Information Exchange Pattern"*)

7.7.5 Information Exchange

Information Exchanges are defined in Service Specifications using defined Message Exchange Patterns [6] (MEP) and features in accordance with WSDL 2.0. All components of WSDL 2.0 need not be defined for every service operation. Only features that are relevant to the particular business domain of a service operation will be specified.

A MEP should be defined for every service operation. The MEP defined within the service description should be considered abstract and not a binding requirement on implementation.

7.7.5.1 Message Exchange Patterns

Only message exchange patterns defined in WSDLADJUNCTS should be used. [7] [8]

7.7.5.2 Features

Characteristics of the exchange patterns are based on the Features defined in WSDL 2.0.

7.7.5.2.1 Multiplicity

Feature	Description
Unicast	This feature defines an operation as having a single target end-point.
Multicast	This feature defines an operation as having zero to many target end-points. A multicast operation can have a single target end-point, but should be able to handle more

[6] Chinnici, R., et al, Web Services Description Language Version 2.0 Part 2 Adjuncts, World Wide Web Consortium, 2006-03-27;http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327/

[7] Outbound message exchange patterns are not explicitly documented in the Standards; however hen implementing, the mirror pattern (i.e. the "other side" of a Broadcast interface) will need to be created.

[8] Booth, D., et al, Web Services Description Language Version 2.0 Part 0: Primer, World Wide Web Consortium, 2006-03-27; http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/

	or fewer end-points as per the service definition.

7.7.5.2.2 Synchronicity

Feature	Description
Synchronous	A synchronous exchange is said to all happen within the same flow of control. This does not imply that only synchronous transports are applicable for synchronous information exchanges provided the synchronous behavior is maintained.
Asynchronous	An asynchronous exchange is said to happen within differing control flows among systems.

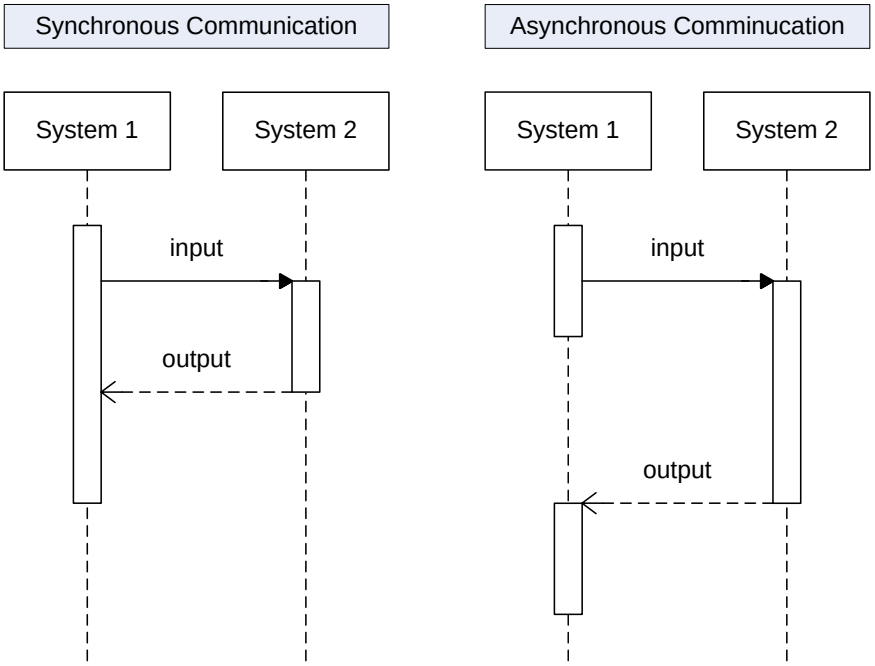

Figure 7-7: Synchronous vs. Asynchronous Communications

7.7.5.2.3 Reliability

Feature	Description
Unreliable	Message delivery will be attempted to the best efforts, but its delivery is not guaranteed.
AtLeastOnce	Message delivery will be made at least once. Duplicate messages may occur.
ExactlyOnce	The message will be delivered once and only once per target.

7.7.5.2.4 Ordering

Feature	Description
Ordered	A interface operation can be defined as being ordered or unordered. This ordering is only in relation to messages associated with the same service. Ordering across services is a choreography that is outside the scope of this document. If the feature is said to be unordered, this feature is omitted. The default behavior is no guaranteed ordering.

7.7.5.3 Usage & Examples

Services are named using a verb and an object. The following verbs are defined for the most common data-exchange service types:

- Broadcast
- Request
- Listen

The following sections define each verb and its associated Features and Message Exchange Pattern.

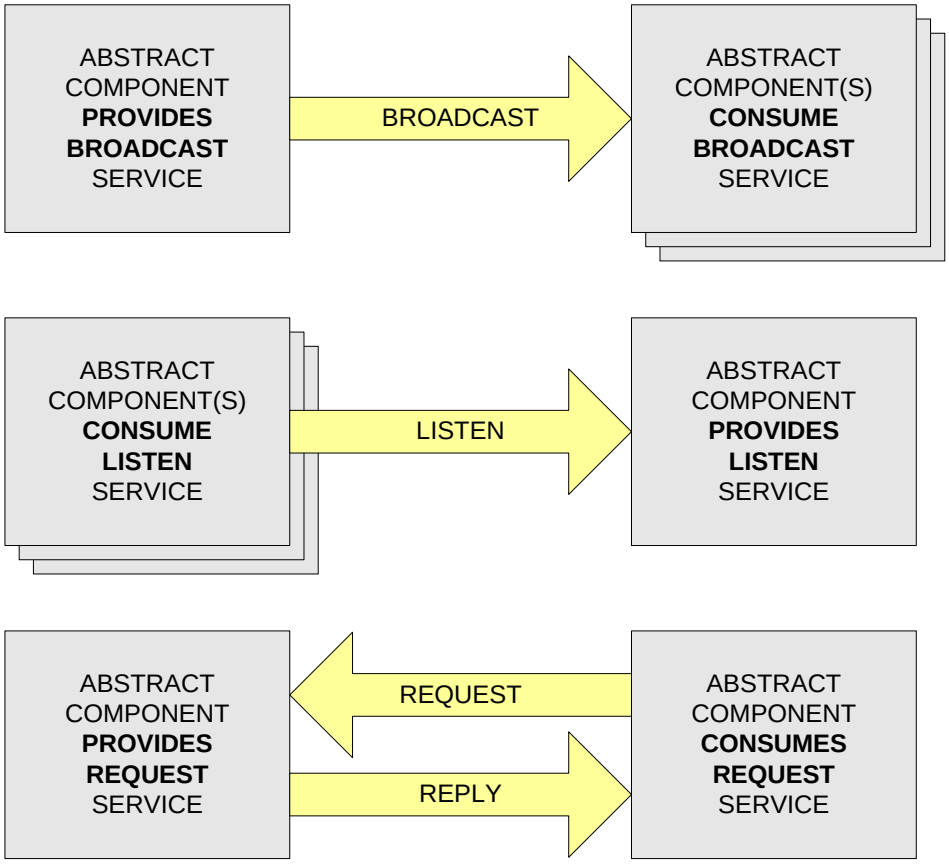

Figure 7-8: Visual representation of defined Exchange Patterns

7.7.6 Status & Error Processing

Error and status conditions should be described in detail within the Abstract Component Description and Service Specification documents. Both status and error conditions should be reported using common formats, data element names, and semantics.

Error conditions in this context are related to the use of services. Each error condition should indicate the circumstances under which a specific error condition occurred. Error conditions should be captured in a persistent form in order to aid administrators in debugging and problem diagnosis.

Status conditions are related to the overall business function of Components. Each status condition should indicate the circumstances under which a specific status condition occurred. Status conditions may be used to report on any situation (e.g. success, failure, informational) and should be captured in a persistent form.

7.8 Applicable Standards

Newcomer, Eric; Lomow, Greg (2005). Understanding SOA with Web Services. Addison Wesley. ISBN 0-321-18086-0.

OASIS Reference SOA Model http://www.oasis-open.org

8. User Interface Architecture

8.1 Overview

The primary goal of the User Interface Architecture is to provide a flexible user interface framework that enables the integration and expansion of multi-vendor solutions into a system which, from an end user perspective, looks and feels as one large application. This implies:

- Common look and feel: the ability to develop end user displays that are consistent, both from a presentation and navigation perspective, regardless of the underlying applications
- Task oriented displays: the ability to develop end user displays based on business processes and work flows, regardless of the origin of the data
- Flexibility: the ability to customize the users displays consistently with the enterprise business practices and style guides
- Display Layout Exchange: the ability to interchange **display layouts** among different vendors and technologies, both in the context of multi-vendor integration and of the migration to future display technologies.
- Multi-vendor console management: the ability to enable displays from multiple vendors to coexist within a shared User Interface Framework.
- Support of security architecture and user authority: the ability to restrict user access to specific applications, data and displays through single sign-on

8.2 Scope

The intent of this User Interface architecture definition is to support the following capabilities:

- **Console Management Standardization** – identify the areas where standards are required in order to integrate information from multiple UI rendering engines into a common console
- **Data Access Standardization** – identify the areas where standards are required in order to integrate data from multiple applications/databases into a common display
- **Display Layout Standardization** – identify the areas where standards are required to allow display layout definitions to be migrated between multiple display editors

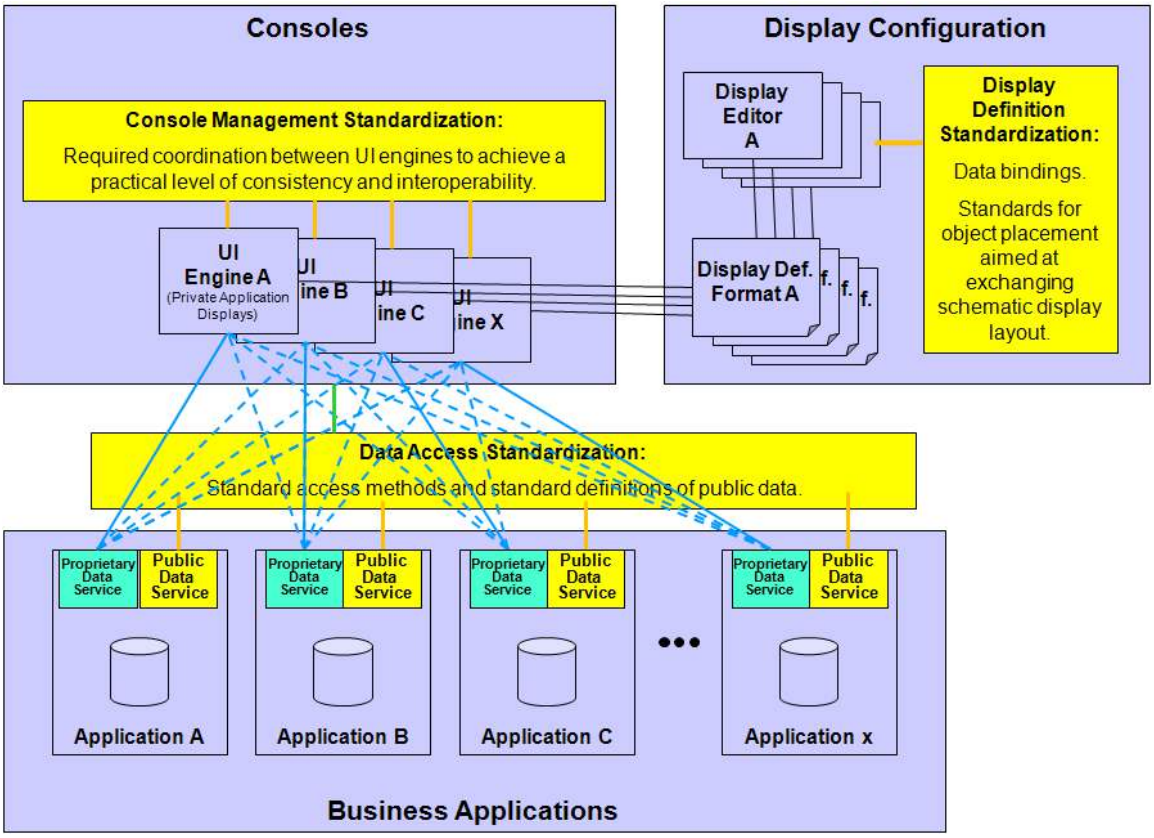

Figure 8-1 User Interface Framework

The User Interface Framework is typically comprised of several components, as shown in Figure 8-1.

- Visualization Environment: Control Room Visualization Environment governs visualization devices that are the primary interface between the Control Room Operator and the grid. Typically, they consist of operator consoles and electronic wall displays.

- Visualization Management, which facilitates user activities that require different UI engines to be active at the same visualization environment at the same time. For more complex user interactions, tasks will also require communication and coordination between these UI components.

- Display Editor, providing the tools necessary to build the Displays Sets. The Display Editor may be integrated into a comprehensive GUI based Power System Modeler, which combines data modeling and display building, and linkages between the two.

- Rendering Engine(s), generating the display content.

- Data Access, providing the access mechanisms for the data to be exchanged between the Data Set Providers and the Rendering Engine(s).

- Display Definitions, including information such as graphical layout, connectivity, data bindings and symbology references.

This document addresses only the architecture related aspects of the User Interface. Specifically, it does not address visualization techniques, style guides, look-and-feel or ergonomics, which are outside of the scope of the document.

The combination of those components into an integrated and fully functional set will be referred to as the User Interface Framework.

Figure 8-2 provides a high level, abstract view of the major components of the targeted UI Framework, along with their key interactions.

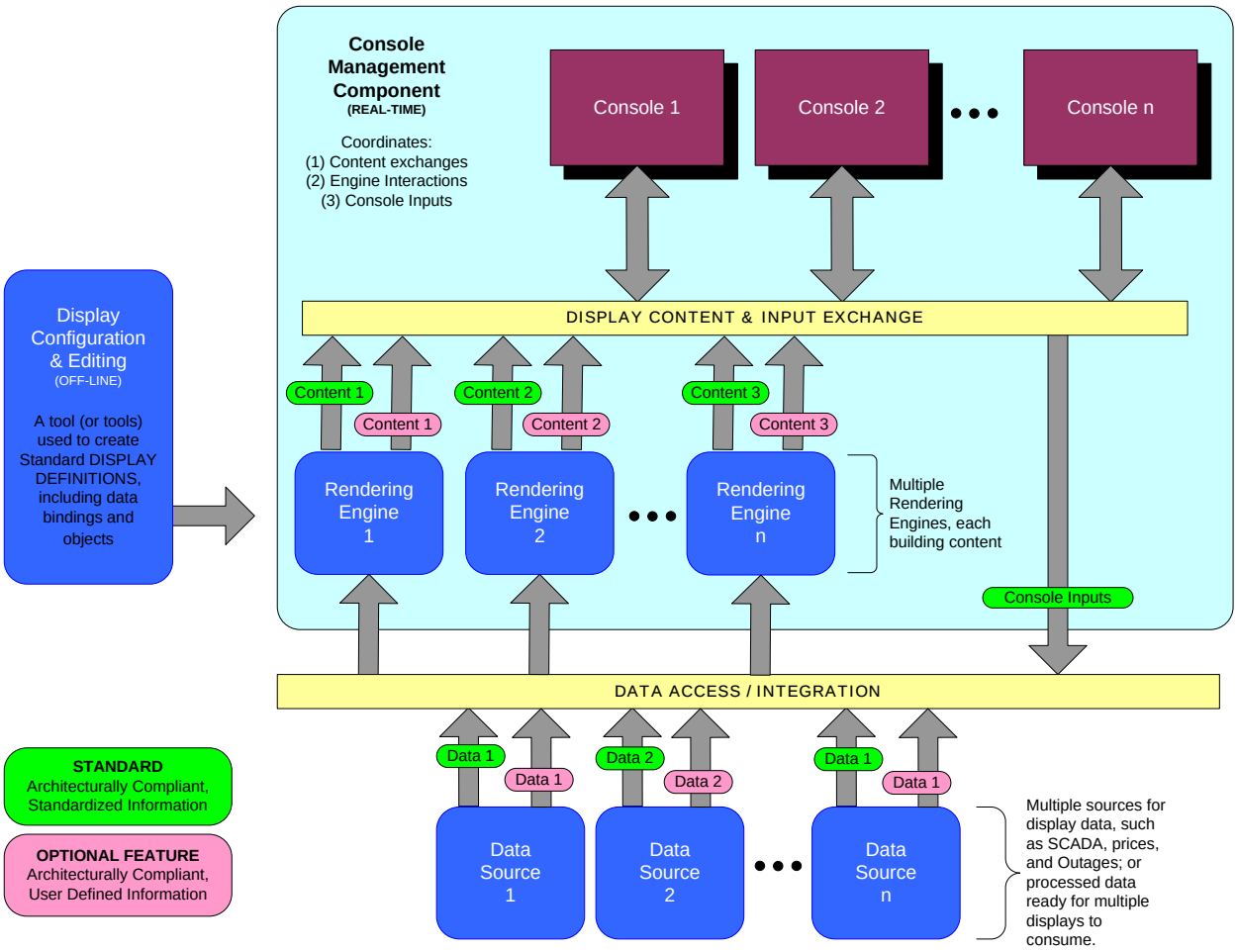

Figure 8-2 User Interface Architecture Overview

The User Interface is represented by a set of Displays which are presented to the User within the Visualization Environment. There are several categories of displays:

Operator Displays: these are used by control room operations personnel to monitor the state of the power grid and take the necessary control actions. These displays include one-line diagrams, geographic displays, contour diagrams, tabular displays, alarms and other special purpose displays. A key objective of the new architecture is that such displays be able to

provide information to the dispatcher, independently of the application(s) that produce the data. In other words, the UI Framework will provide the capability to generate displays that are workflow oriented and/or task centric, as opposed to the current generation of displays which are application centric (Closely integrated with the application that provides the data).

Engineer Displays: these are non operational displays used by engineering and maintenance personnel. They are typically application and vendor specific, and less critical in the achievement of the above business objectives. Consequently, the technical requirements enumerated above do not apply to this category of displays, although it would be preferable that they do on the long term. Special consideration should be made for displays associated with the overall computer system operation.

Management Displays: these dashboard type displays would be used by management or non-operator/engineer personnel. These displays may also be available external to the operational environment.

It is recognized that it is not practical on the short term for vendors to replace their User Interface with a vendor/application independent User Interface. During a transition period to be defined as part of a jointly defined WG D2.24 Roadmap, the vendor User Interface can work concurrently with a vendor/application independent visualization that is decoupled from proprietary datasets and applications.

This necessitates the coexistence of two methods of visualization:

- A Task based UI, which is vendor independent, and based on a shared UI Framework. Such framework can be either a third-party UI Framework, or one of the system suppliers UI Framework.

- An Application based UI, which provides legacy access to the application data for operation and/or maintenance purposes, using the specific vendor proprietary technologies and display formats.

However, the vendor specific architecture implementation should enable fast access, as dictated by performance requirements, to the data produced by one or more applications, in order to incorporate them within the task or workflow oriented displays. Conversely, the UI Framework will enable application suppliers to generate easily new displays using the UI Framework data access mechanisms and tool boxes.

The UI Architecture Framework should provide a Visualization Management capability that enables displays from different vendors to share the same physical space on the Visualization Environment.

8.3 Architecture Description

The User Interface Architecture is designed using the following principles:

- Decoupling of the presentation from the application: this implies that it will be possible to leverage data sets produced by various applications through a shared User Interface framework.

- Standardization of Display Layout Exchange Format: this requirement applies primarily to one-line diagrams and tabular displays. It implies that it will be possible to exchange display layouts between different vendor visualization solutions.

- CIM based information models: exchange of information related to visualization leverages the CIM.

- CIM Support: a CIM based Power System Modeler can be used to generate display layouts for visualization.

8.3.1 Rendering Engines

Graphic information is produced on the visualization devices by means of one or several Rendering Engines.

"Rendering" is the process of generating an image from a model, by means of computer programs. The model is a description of graphical objects in a strictly defined language or data structure. In the context of EMS/MMS, the term "Rendering Engine" will apply to any graphic engine that converts an abstract graphic specification, and related dynamic information, into a set of rasters on the display device, or "Client".

Clients can be categorized based on the amount of local graphic processing.

At one end of the spectrum, a "Rich Client" can leverage advanced visualization techniques in real-time, including high performance animation of graphical objects. This capability, however, is provided at the expense of portability, interoperability and remote access capability.

A "Thin Client" is at the other end of the spectrum. The main advantage of the thin client is that displays can be produced remotely on lean platforms, leveraging widely available technologies such as Web Browsers. These techniques are highly interoperable, since the displays can be generated locally, independently of the location and underlying technology associated with the application generating the data.

Both types of rendering engines are useful: Rich Clients in the control room, in order to provide control room operators with the most appropriate visualization tools with an acceptable performance, and Thin Clients in order to provide remote users with relevant, albeit more limited information.

The Rendering Engine also has functionality related to security. Any operational data that requires interactive access should be accessible via a Rendering Engine that has proper authentication and authority restrictions.

8.3.2 Display Definitions

Display Definitions are used to define a variety of UI components: one-line diagrams, forms, reports, dashboards, dialogs and other visual elements. Display definitions may use proprietary formats. However, there is a very compelling need for the introduction of Standard Display Layout Exchange Formats. This is for several key reasons:

- The ability to exchange displays from multiple vendors within a utility specific implementation.

- The ability to generate the same display through multiple rendering engines, i.e. on a rich client for control room operation and on thin clients for remote applications.

- The ability to migrate displays to future technologies and vendors, thus preserving the substantial utility investment in the display definitions.

It is not the intent of this requirement, at least on the short term, to replace the proprietary Display Definition formats. Likewise, it is not the intent of this requirement to introduce through the standardization process unnecessary constraints that would limit the specific vendor implementations. It is the intent, however, to drive toward a standardized display layout exchange format specification that will enable the generation of functionally equivalent displays.

The recommended approach is:

- Extension of the Common Information Model as needed to define profiles for the exchange of display layouts.

- Adoption of a common language for the display specifications (either CIM XML or XML Schema).

- Import/export capabilities between the standard Display Layout Exchange Formats and Display Editors.

8.3.3 Display Editing

Display Editing is an integral part of the power system modeling process. The Display Editor is a tool that is used to create and maintain Display Definitions. The Display Editor may be a stand alone tool, or embedded into a Power System Modeler.

The objective of the Power System Modeler/Display Editor is to provide the ability to develop the power system model data and associated display representation/connectivity through a common integrated process. Minimum functionality includes the ability to:

- Auto-generate display layouts from a given power system model and/or graphically edit the power system model.

- Dynamically link display objects to their associated data sets (Data Binding).

- Enable exchange of Display Layouts using the Standard Display Layout Exchange Format (See Figure 8-3).

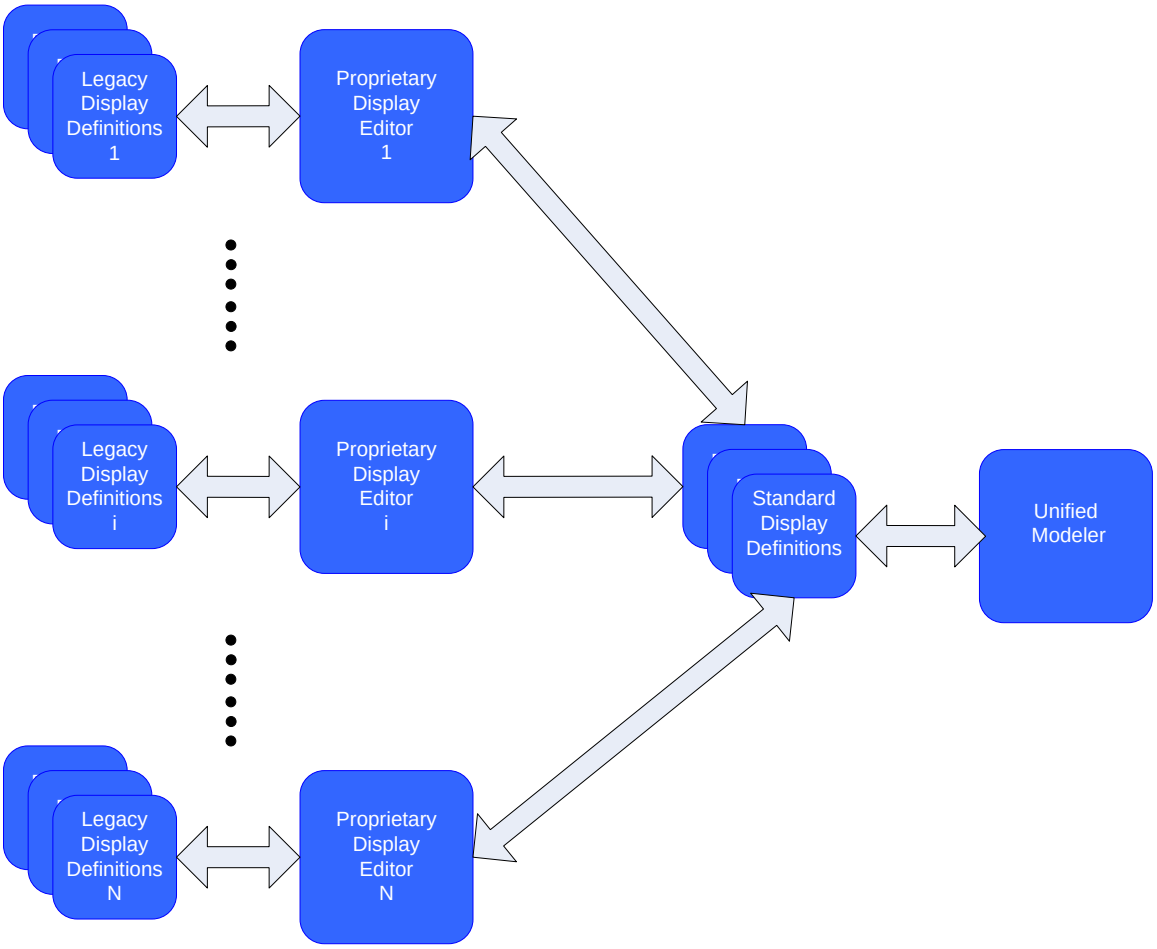

Figure 8-3 Display Editing

8.3.4 Data Access

At the heart of the UI Framework is the UI Data Access. The objective is to abstract the data from the displays, providing the capability of building displays agnostic to the source of the data. In particular, Data Access provides for easy integration of data generated by any application to present data/information, and conversely, the ability to upgrade the user displays without changing the application code or data sets.

The Information Architecture provides the mechanisms to request and obtain data to be displayed from the various data sources. The UI will have the same data access capability as any other Consumer as defined in the Information Architecture.

A key feature of Data Sets is a standard, CIM-based agreement about the structure and content. This allows a consuming Rendering Engine to know precisely how to ask for data, no matter who supplied the business component that produced the data. Any display that uses exclusively Data Sets, no matter how many different producers are involved, can be specified in the same way for any system. Applications may need data not available in Data Sets.

Therefore, there needs to be a process in place (e.g. through the CIM) to extend Data Sets with application specific data where necessary.

The requirement for proprietary data access is that the supplier of the data makes the data "open" by a) not placing legal restrictions on access, and b) creating a well-documented supported access method.

8.3.5 Visualization Environment

The Visualization Environment is the primary interface between the Control Room Operator and the grid, and currently includes Operator Consoles and Electronic Wallboards. In the future, new visualization devices will emerge and complement the array of tools available to the Operator. Examples of such tools include Touch Tables and 3D goggles. Thus, the UI Architecture Framework will need to be flexible enough to be decoupled from the physical devices and accommodate future visualization technologies.

Operator Consoles: Operator Consoles are associated with specific control room functions. A key security requirement is the "Single Sign On" feature. This means that the Operator needs to sign on only once at the console to access all the displays and functions associated with his role and authorization level. That includes bringing automatically displays that the Operator will have customized to his/her own preferences. This concept is illustrated on Figure 8-4. Another important requirement is the ability for the Operator to interact with the system through task based displays, which provide him with all the information needed to perform his/her tasks with minimum navigation between displays.

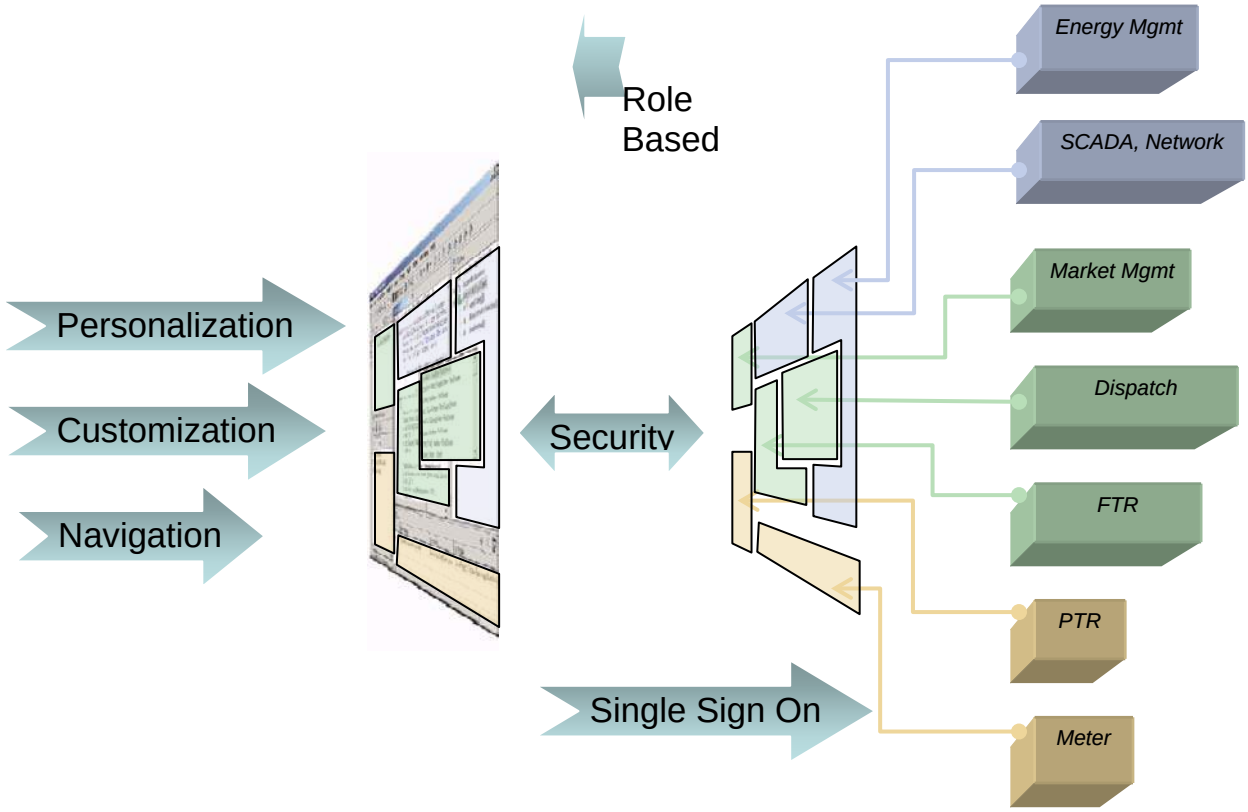

Figure 8-4 Common User Interface

Wallboards have been traditionally designed in order to provide a system overview. Typically implemented through an array of mosaic tiles silk screened with a one-line diagram representation of the system, they provide System Operators with a shared overview of the power grid. They are/were often complemented with a Projector able to reproduce selected Operator Displays. As the cost of technology comes down, hardwired wallboards are increasingly being replaced with Electronic Wallboards, which provide an expanded workspace on a wall of large screen seamless displays. The Electronic Wallboard provides a great amount of flexibility, as it enables the selection of multiple, concurrent views, fit to the situation at hand. Such views may include traditional one line diagrams or geographic displays with various types of overlays such as contour diagrams or dynamic weather maps. The many potential uses of the Electronic Wallboard are outside of the scope of this document.

8.3.6 Visualization Management

Visualization Management provides the capability for different UI Rendering Engines to share the same visual environment. Visualization Management should recognize the following functional needs:

- Visualization Layout: Visual placement and management between different Rendering Engines
- Navigation: Cross window and cross display navigation

- Session Management and Security: Support for single sign-on, sharing of user session information across displays, and secure communication between UI applications and business services.

- Inter-display communication: Communications that facilitate common user interactions among displays.

Further work is needed to define detailed requirements.

8.4 Applicable Standards

The recommendations in above section related to Display Layout Exchange are contrary to IEC 61970-453 ‘CIM-Based Graphics Exchange’ and IEC 61970-553 ‘CIM-Based Graphics Exchange’. The recommendations of this section are based upon IEC 61970-552 ‘CIM XML Model Exchange Format.’

9. Cyber Security & Network Architectures

9.1 Overview

Electric industry EMS/MMS systems and control systems increasingly rely on commercial information technologies such as Ethernet, TCP/IP, and the Internet for both critical and non-critical communications. The use of standard IT infrastructures and common communications protocols has made integration of external interfaces much easier with increased business efficiency and reduced operational costs. However, multi-network integration strategies have resulted in significantly less isolation from the outside world for vital SCADA and power system control networks. However, advances in communication network integration create security and vulnerability issues especially for legacy systems not designed to operate in an open network environment.

The objectives of the cyber security and network architectures are to ensure:

- **Availability** – ensure systems and data are available to authorized users when they need it.
- **Integrity** – ensure data is not tampered or altered by unauthorized users.
- **Confidentiality** – ensure access only to data for which the user is permitted.

9.2 Scope

This document addresses the architecture related aspects of information systems security and the security aspects of communications networks. It is not the intention of this section to address network architectures in general terms but to focus on network architectures from a security perspective. The concepts and principles presented in this section are organized into the following topics.

- **Communications network segmentation:** Systems will reside in networks with perimeters clearly defined by security zones. Security zones allow enterprises to divide the physical network into a series of virtual sections, so that various levels of trust and security policies can be established. Each zone may require its own set of domain/DNS appliances (this raises the cost but lowers the maintenance efforts and increases reliability). Network segments reside within security zones.
- **Perimeter protection:** Perimeter protection is the first defense that addresses the levels of protection, and provides the foundation for the communication specifics between security zones.
- **System and communication protection:** This area addresses the security of the communication interfaces and protocols (Refer to IEC 62351), as well as the services and components built-in security. It also addresses system hardening and the use and management of digital certificates and digital keys.

- **User security:** User security addresses the ability to define access privileges via roles, authentication, and authorization services based on user accounts and authorities. Identity Management and Authority Management are the abstract systems that provide authentication of user credentials and authorities across multiple components and security zones. Shared end user accounts should be forbidden. Multi-factor authentication is required for interactive access from outside the security perimeter.

- **Centralized audit and monitoring:** This area addresses the centralized logging, auditing, and correlation functionality for identifying policy violations and fraudulent activities; for incident report and response; for forensic analysis; and for government and industry regulation compliance.

- **Change management and system integrity**: This area addresses the ability of automatic auditing changes and configurations for system integrity and policy/regulation compliance.

- **Shared Storage:** Databases or Storage Area Networks (SANs) that are shared require the application of virtual SANS (vSANs) and zoning techniques.

- **Virtualization:** Virtualization requires careful consideration. Each hardware/Operating System has different control mechanisms, but allows sharing of hardware between virtual guests/partitions. A control program that controls this should be reviewed for security consideration. Each virtual environment should follow all the security requirements. Redundancy and reliability of both the hardware and the networking should be considered when hardware is shared. Virtualized systems should reside within an individual security zone.

The combination of the above areas will constitute the Cyber Security and Network Architectures. The goal is to provide an architecture that supports all of the energy management and market management business requirements and business processes in an open but secured network environment, where the compliance to NERC CIP 002 through CIP 009 and other general government or industry cyber security regulations is supported intrinsically.

System protection includes both physical and cyber protection. This security architecture addresses only cyber protection. Physical protection plays an important role in the defense in depth strategy and is required around critical cyber assets as defined in NERC CIP 003. Application security is just as important as network security to the system. Most common cyber attacks, such as buffer overflow, data insertion and injection and Denial of Service, take advantage of the defects in application software. These vulnerabilities are particular threatening because normal network access control checks, like firewalls, IDS/IPS, and authentication can be bypassed. Secure systems are quality systems. A secure program needs to be a robust program. It is of paramount importance that code is designed and built with security as a prime feature, not an afterthought. Vendors (and utilities) that develop in-house solutions should have a security development process in place including secure coding requirements and code reviews.

9.3 Architecture Description

This section describes the key design principals and how the components work together to ensure a cohesive secure system in a networked environment. It also provides guidelines for how to use information standards in implementation, deployment and maintenance that conforms to this architecture.

9.3.1 Key Security Design Principles

The security principles listed in this section are used in the design of this security architecture. These security principles should be used in the low level design, implementation, deployment, operation, maintenance and disposal of a system throughout its life cycle. They can also be used in affirming and confirming the security posture of legacy systems.

- **Defense in depth** represents the strategy of using multiple computer security techniques to help mitigate the risk of one component of the defense being compromised or circumvented. The principle of defense in depth is that where one control could be reasonable, more controls that approach risks in different fashions are better. Multiple layers of defenses can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur.

- **Minimize attach surface** addresses the issue that the more points of entry into a system provide more avenues for an attacker to find a weakness for penetrating the system. Minimize attack surface includes actions such as removal of unnecessary services and applications, shutdown of unused network ports, removal of unused accounts, and other system hardening practices.

- **Keep security simple** reduces attack surfaces and chances of making mistakes. It increases the performance of a system and decreases the cost of operations. A simple security solution is more likely to be deployed and used than a complex one. Simplicity makes configuration and maintenance tasks less expensive and less likely to accidentally introduce new attack surfaces.

- **Separation of duties** is the concept of having more than one person required to complete a task, so that a deliberate fraud is more difficult to occur because it requires collusion of two or more individuals or parties. For example, Operators must not be able to set their own authorities (area of responsibilities). Five general categories of functions need to be separated.

    - Authorization function: IT governance board or manager who decides the roles and the functionalities of the roles;

    - Design and development: developers;

    - Deployment: IT department and/or EMS/MMS engineers;

    - Daily operation: operators who are assigned to a specific role or multiple roles that doesn’t violate the principle of SoD;

    - Supervisory: EMS/MMS administrators.

- **Least privilege** is widely recognized as an important design consideration in enhancing the protection of data and functionality from faults and malicious behaviors. It recommends that in a particular abstraction layer of a computing environment, every module (including a process and a user) should be able to access only such information and resources that are necessary to perform their assigned business processes, but not more. Resources permissions encompass CPU limits, memory, network, and file permissions.

- **Security through obscurity** is a weak security control that nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems must not be reliant upon keeping details hidden. Always assume that an attacker has access to all source code and all designs. Even if it is not true, it is trivially easy for an attacker to determine obscure information.

- **Assume external systems and services are insecure.** Network segmentation provides a set of well defined security zones for different levels of trust. An external system is the one that is not under your control. An external system should be considered non-secure until it has been deemed "trusted". This security principle also applies to external services in Service Oriented Architecture.

- **Establish secure defaults** so that the factory default installation will be secure. It should be up to the organization to reduce the security levels based on their security policies if needed. For example, password complexity and aging should be defined to the IT standard of strong password and to industry standards. A particular deployment may decide to reduce the password complexity requirement for ease of use, but they are aware of the risk and willing to accept the risk. Default vendor passwords should be changed.

Use positive security model (white-list) to define what is allowed and reject everything else. This is contrasted with the negative security model, a.k.a. blacklist, which defines what is disallowed while implicitly allowing everything else. In the access control area, the positive model is to deny access to everything, and only allow access to specific authorized users, resources or functions. Network firewall rules usually take the positive model approach. The benefit of using a positive model is that new unanticipated attacks will be prevented. Adopting the negative model means that you'll never be quite sure that you've addressed everything. You'll also end up with a long list of negative signatures to block that has to be maintained. The principle of use positive security model is consistent with the principle of keep security simple.

9.3.2 Network Segmentation

New systems should reside in communication networks with perimeters clearly defined by security zones. Security zones are usually implemented as security domains. They can be grouped into security levels and color-coded in this security architecture. Each system should reside within a prescribed security zone at a specific level. Reasons for segmentation include reliability and separating types of external traffic such as User Interface, Historian, Inter-Control Center Communications, RTU Communications, etc. New systems should be configurable in the way security zones/network segments are defined.

A Security Domain is a logical group of functionally related IT resources, including computers, users, network infrastructure etc, which requires a similar level of strength and assurance of IT security. Attributes of security domains include:

- A boundary that defines which systems are “inside” the domain and which systems are “outside” the domain;
- A uniform security policy and under a single administrative control;
- All inter-domain traffic is mediated by Policy Enforcement Points (PEP);
- A single security domain does not normally exist in multiple geographic locations;
- Domains do not overlap.

9.3.2.1 Security Zones

Security zones are divided into levels and color-coded. Each level reflects common or shared security attributes. Multiple security domains can reside in the same level. Levels generally reflect varying levels of risk.

- **Level 0 - Red Zone: External Domains**

Infrastructures that are not under direct or delegated control should be placed in Level 0 - Red Zone. All external systems in the Red Zone should be considered non-secure until they have been deemed “trusted”. These include:

- The Internet;
- Business partner’s network;
- Public switch telephone network (PSTN).

- **Level 1 - Orange Zone: External Gateway/Proxy Domains**

Infrastructures that are connected to a sub-network of a perimeter PEP are placed in Level 1 - Orange Zone also commonly known as a Corporate De-Militarized Zone (DMZ). It contains systems that provide a secure proxied “stepping stone” between Level 0<->Level 2 and Level 1<-> Level 3 domains. These secured systems can include:

- DNS servers;
- Exchange servers;
- Corporate web servers;
- Corporate authentication servers.

● Level 2 - Yellow Zone: Internal Domains

Infrastructures that are under direct or delegated control of and that ARE NOT DIRECTLY related to managing, monitoring or controlling elements of electricity grid should be placed in the Yellow Zone. The infrastructures in the Yellow Zone can include:

Corporate business network, end-user systems, file & print servers, HR & Finance servers and associated network infrastructure;

Internal IT administrative systems related to the IT management/administration of IT infrastructure in Levels 1 & 2 and associated network infrastructure;

Market Management Systems (MMS) that are directly related to the running of the market and associated network infrastructure but do not send regulation signals out to other companies.

● Level 3 - Green Zone: Internal Gateway/Proxy Domains

Infrastructures that are connected to a sub-network of an internal PEP are placed in the Green Zone. The Green Zone is also commonly known as a DMZ. It contains systems that provide a secure proxied "stepping stone" between Level 2<->Level 4 and Level 1<-> Level 3 domains. These secured systems can include:

- ICCP servers;
- Historian servers;
- UI web servers;
- Authentication servers.

● Level 4 - Blue Zone: Sanctum Domains

Infrastructures that are under direct or delegated control of and that ARE DIRECTLY related to managing, monitoring, or controlling elements of the electricity grid or market should be placed in the Blue Zone. The infrastructures in the Blue Zone could include:

- Power System Network Monitoring and Control systems, including EMS servers & clients, Frequency & Voltage monitoring systems, and associated network infrastructure;
- Market Management Systems that issue regulation signals to other companies;
- Substation infrastructure including local Substation Control Systems, metering systems, building services systems, and associated network infrastructure;
- Administrative systems related to the management/administration of infrastructures in Levels 3 & Level 4, and the associated network infrastructure.

9.3.2.2 Access Points between Security Zones

The design principles for application/system integration with security domains include:

- A system should not be a member of a domain if the member has a higher protection requirement for confidentiality or mission assurance than is offered by the domain;

- Information flows should be initiated/pushed from higher level domains to lower domains, via some form of proxy server if necessary, as opposed to being initiated/pulled directly from lower security domains;

- Applications/systems are NOT permitted to initiate direct connections where the destination end point is in a Level 2 or 4 domain AND the source end point is more than one level below the destination end point;

- Applications/systems are permitted to initiate direct connections where the destination end point is in a security domain that is no more than two levels below the source end point.

- Controls should be designed to allow each domain to protect itself where possible – i.e. reliance on transitive trust should be minimized;

- Based on the principle of least privilege, a domain perimeter should allow PEP rules/policy to be established such that they support the most specific capability to fulfill the required functionality.

Figure 9-1 presents an example of a logical network partitioning for modern EMS Systems operating in an environment with a primary control center and a remote backup control center.

The production EMS servers, UIs, and other systems that directly related to managing, monitoring and controlling the electricity grid should reside in the Level 4 Blue Zone. A utility may have multiple control centers and backup sites. Security domains of primary control centers and backup centers should be carefully designed based on the design principles listed above. The two EMS control centers are usually two sets of duplicated systems. They are in two separate level 4 domains with identical security requirements.

SCADA data acquisition servers for receiving data from substations and sending commands to substations located in the Level 4 Blue Zone, but in a separate domain from production EMS servers.

Servers for communicating with neighboring SCADA systems should be in the Level 3 Green Zone, also called a DMZ. The Green Zone provides a shield to the highly secured production EMS domains from remote connections and/or less secured servers. Historic data servers and databases should reside within the Level 3 Green Zone. Data flows from EMS servers in the Blue Zone (higher security level) to historic data servers or historian databases in the Green Zone (lower security level). The historic data servers or databases feed near real-time data to EMS servers which sit in the Level 1 Orange Zone.

The Level 2 Yellow Zone is for internal corporate business network. Business related application servers, such as HR and finance servers, file and print servers, and end-user systems reside in Level 2 domains.

The functionalities of EMS servers usually are used by both internal users and business partners who connect from the internet or via VPN. EMS servers reside in the Level 1 Orange Zone, but in a separate DMZ from other internal corporate business DMZs. The remote VPN connections from business partners should be restricted to the EMS DMZ. EMS DMZ does not have a direct connection with real-time SCADA systems for minimizing potential disruption from the internet. It can talk to Control System DMZs so the historian DMZ can feed data into the EMS DMZ.

Corporate web servers, email servers and other internet facing systems should be put in the Level 1 Orange Zone, also called a corporate DMZ. There should not be any direct communication between Level 1 (Corporate DMZ) and Level 4 (Control System Network).

It is recommended that EMS Product Development System (PDS), Quality Assurance (QA) systems and training systems be put in a Level 4 Blue Zone, but in separate zones/domains from the production domains. PDS, QA and training systems contain critical Control System cyber assets. Accordingly these systems need the highest security measures, and separation from production environment.

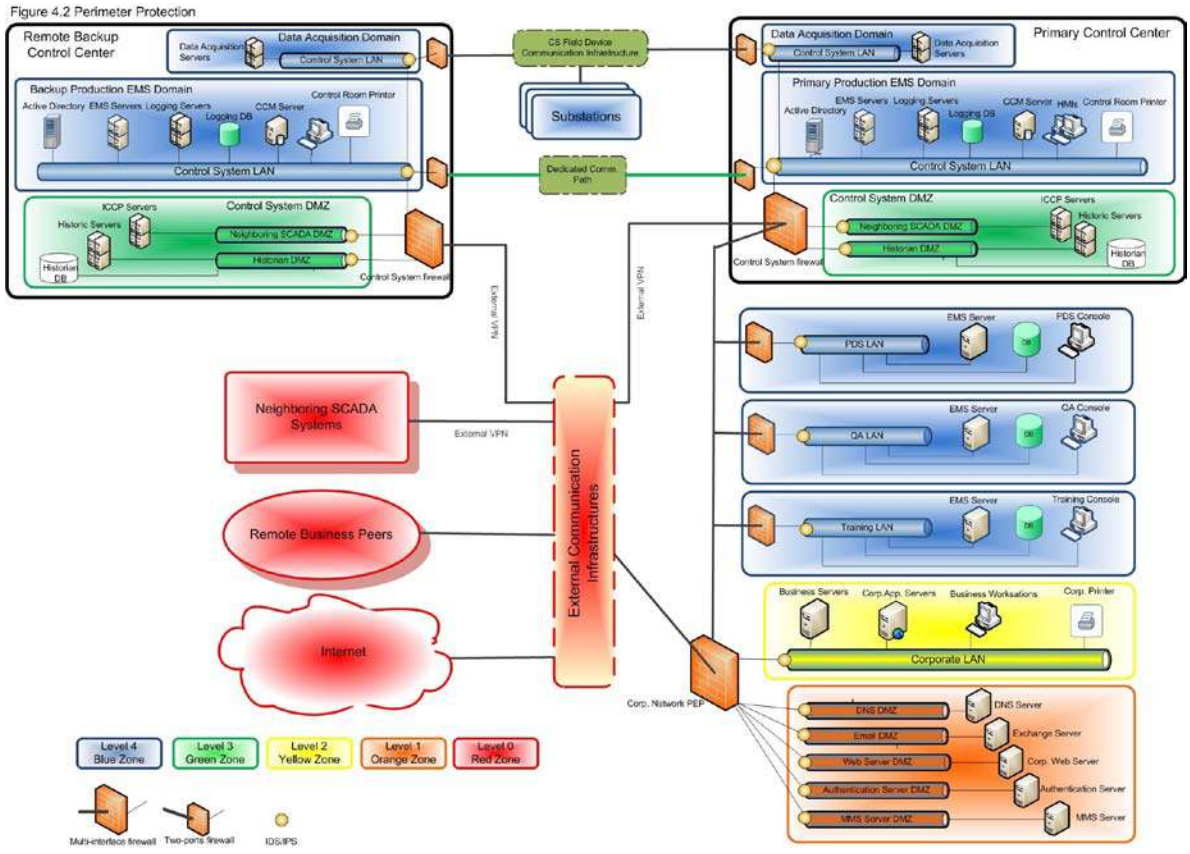

Figure 9-1: Network Segmentation

9.3.3 Perimeter Protection

Network segmentation provides clearly defined perimeters. Perimeter protection is critical to mitigate cyber attacks at the frontline. Policy Enforcement Points (PEP) should be placed at each domain perimeter. PEP rules/policies should be defined to the most specific requirements such that they support the most specific capability to fulfill the required functionality, adhering to the principle of least privilege. When considering perimeters, it is critical to conduct a comprehensive review during implementation. Every point of entry through the perimeter should be considered and protected accordingly.

9.3.3.1 Definitions

For the purpose of addressing perimeter protection in this security architecture, this section gives brief definitions of the PEP and PEP devices that are security related and used in the architecture.

Policy Enforcement Point (PEP). A PEP is a network device that enforces policy decisions and can alter the traffic flow from one network to another. Some examples of PEP are routers, switches, firewalls, Intrusion Detection System (IDS), Intrusion Prevention Systems (IPS), and VPN systems. Simply put, a PEP is the place where the determination is made that a policy needs to be enforced.

Firewall. A firewall is a mechanism used to control and monitor traffic to and from a network for the purpose of protecting systems and applications on the network. It compares the traffic passing through it to a predefined security criteria or policy, discarding messages that do not meet the policy's requirements. In effect, it is a filter blocking unwanted network traffic and placing limitations on the amount and type of communication that occurs between a protected network and other networks.

There are two basic types of firewalls, Network-Based and Host-Based. Network-Based firewall is usually a separate hardware or hardware/software unit. It is typically installed on the network perimeter, and provides the most secure solution for the separation of security domains. A Host-Based firewall is a software program installed on a server or a workstation. While the rule/policies can be configured more specific to a Host-Based firewall than a Network-Based firewall, Host-Based firewalls can have more impact on system performance.

Intrusion Detection System (IDS) and Intrusion Prevention Systems (IPS)**.** Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices.

An intrusion detection system (IDS) is software that automates the intrusion detection process. An intrusion prevention system (IPS) is software that has all the capabilities of an intrusion detection system and can also attempt to stop possible incidents. IPS is recommended for system perimeter protection to only block traffic between zones that can be blocked without impacting the reliability function of the EMS/MMS.

IDS/IPS use many methodologies to detect incidents. Common detection methodologies are signature-based, anomaly-based, and stateful protocol analysis. Most IDS/IPS technologies use

multiple detection methodologies, either separately or integrated, to provide more broad and accurate detection. IDS/IPS is categorized as Network-Based, Host-Based, wireless, and network behavior analysis (NBA) system.

Malware. Malware is software designed to infiltrate or damage a computer system without the owner's consent. It includes viruses, worms, Trojan horses, spyware and other malicious and unwanted software.

Anti-Virus software. Antivirus software is a computer program that attempts to identify, thwart and eliminate malware.

9.3.3.2 The Goals of Perimeter Protection

Routers, firewalls and IDS/IPS are widely used for perimeter protection. They are effective PEPs that separate different domains, redirect network traffic, enforce security rules/policies, and detect/prevent unsolicited network and network flooding traffic. See Appendix x for a sample perimeter protection strategy.

The overall security goals of system perimeter protection are as follows:

Prohibit direct connection from a Red Zone to a Blue Zone. Connection from the internet and untrusted entities are strictly prohibited to the SCADA and other systems that directly manage the electricity grid.

Prohibit direct connection from a Yellow Zone and an Orange Zone to a Blue Zone. Even though corporate internal networks are controlled by the corporate IT department, many business applications and systems installed on the corporate network are common and easy targets of various cyber attacks. Unnecessary traffic between a Yellow Zone and a Blue Zone could not only congest and corrupt the system network but also make the system more vulnerable to cyber attacks.

Restrict access from a business network to a Green Zone. Typical access from a business network to a Green Zone is EMS servers in the Orange Zone accessing SCADA historian servers in the Green Zone. Access control and protocols should be enforced by PEP at the boundary. For example, telnet and FTP protocol should be prohibited.

Restrict access between a Green Zone and a Blue Zone. Connections should be initiated from a Blue Zone to a Green Zone. Data saved to a historian database should be pushed from the production EMS server to the historian server, not pulled by the historian server. Data servers, such as ICCP servers that connect to neighboring SCADA systems, are placed in a Green Zone. The connection between EMS servers and data acquisition servers should be initiated by EMS servers from a Blue Zone; mutual authentication should be in place; EMS servers should pull the measuring data from data acquisition servers; the firewall rule should filter out all inbound traffic, except valid measuring data and expected responses. The firewall should prohibit all outbound traffic to the green zone, except valid control commands and status requests.

Restrict access between domains in a Blue Zone. Domains at a Blue Zone have the same security requirements. The main reason of separating the primary production domain and the backup production domain is for overall system availability. Communications between the two domains should be allowed, but access control should be strictly enforced. Protocol

enforcement rules should be in place to reduce network traffic. The specific PEP rules/policies between Blue Zones depend on the domain functionality. For example, the firewall rules between the primary production domain and the backup production domain should be different from the firewall rules between the primary production domain and the QA domain for obvious reasons.

Detect and prevent unsolicited network events. Network-Based IDS/IPS should be deployed at domain perimeters to detect and prevent potential incidents. During the network design process, much thought should be put into the IDS/IPS architecture. Single or multiple types of IDS/IPS technologies should be considered to achieve a comprehensive and accurate detection and prevention of malicious activity, while providing maximum efficiency and minimum impact on system performance. Refer to NIST publication 800-94: Guide to Intrusion Detection and Prevention Systems (IDPS) for recommendations on designing, implementing, configuring, securing, monitoring, and maintaining IDS/IPS.

Detect and eliminate malware. Antivirus software is categorized as signature-based and/or behavioral-based. The traditional signature-based technology has worked effectively for detecting, quarantining or deleting known malwares. The proactive behavior-based technology offers better protection against zero-day attacks and other threats based on characteristics, not signatures. Malware detecting and elimination can be network-based and host-based. A combination of antivirus technologies and perimeter malware detection and elimination should be deployed in a system to achieve maximum protection, with a minimum performance impact on the system.

9.3.4 System and Communication Protection

This section addresses the security of the communication interfaces and protocols, as well as the services and components built-in security. It also addresses system hardening.

The integration of EMS Systems into enterprise networks raises many challenges on data security and system stability. Protecting critical infrastructure data and company trading secrets become increasingly important and difficult. The security goals of system and communication protection are:

- Provide data confidentiality and data integrity for data in transit
- Communication protection plays extremely important role to ensure data confidentiality and data integrity while data is transferred between domains over public/shared network.
- Provide data confidentiality and data integrity for data in process and in storage
- System protection and services/components built-in security offer data integrity and confidentiality when data is being processed and saved in storage.
- Strengthen the defense ability of perimeter protection

Perimeter protection is effective only if communication between domains and systems are carefully designed to be firewall/IDS/IPS friendly. Firewall/IDS/IPS friendly means using communication protocols that are industry standards and supported by the firewalls and IDS/IPS manufactures/vendors. A system is as secure as the weakest link. It also means keeping the communication path simple and well defined.

9.3.5 User Security

This section addresses the ability to define access privileges based on user profiles that define the user's role and authority. The architecture supports strong authentication and authorization using the latest technologies, and is scalable to different deployment environments. It is extensible for adapting to future access control technologies.

This architecture does not specifically address service-to-service authentication nor does this architecture establish specific requirements for issuing authentication credentials to users when they are used in the systems. User credential issuing and management should conform to NERC CIP or applicable standard and corporate security policies. Network Authentication (machine to machine authentication) should use 802.1x network authentication.

9.3.5.1 Definitions

Authentication. Authentication is the process of determining whether someone or something is, in fact, who or what it is declared to be.

Authentication Factors. Authentication systems are often categorized by the number of factors that they incorporate. The three factors often considered as the cornerstone of authentication are:

Something you know (for example, a password)

Something you have (for example, an ID badge or a cryptographic key)

Something you are (for example, a voice print or other biometric)

Strong Authentication. Authentication systems that incorporate two or more factors are called strong authentication. Systems that incorporate all three factors are stronger than systems that only incorporate one or two factors.

Single Sign On. Single Sign On (SSO) is a method of access control that enables a user to authenticate once and gain access to the resources of multiple software systems.

Authorization. Authorization is the process of granting access privileges to a user, program, or process.

Role-Based Authorization. In a role-based authorization system, also known as Role-Based Access Control (RBAC), users are assigned one or more predefined roles. These roles then determine the user's privileges to the requested resources.

9.3.5.2 The Access Control Framework

The Access Control Framework is composed of three framework services: Login, Authentication and Authorization. Each is defined as a service interface and can have multiple implementations. The three framework services work together to acquire user credential, verify the credential, and grant the privileges to the user. The implementations of the service interfaces can be called by other services to perform the functionality it is design for, or can be

plugged into the Integration Layer to provide login, authentication and authorization service to other components deployed on the Integration Layer.

The Login Service exposes a specific interface that is invoked for retrieving credential(s) from the user. The service is expected to display a login dialog (if one is desired) that is appropriate for the deployment environment, and to gather and return the user's credentials. The Access Control Service orchestrates the following:

- Use Login Service to gather user credential(s);
- Use Authentication Service to establish the validity of the user and create an *opaque token*;
- Use the Authorization Service to load Authorization information.

Figure 9-2 illustrates the relationships between the key components of the Access Control Framework. Of note are:

Login Service: This service could implement 3-factor, 2-factor, biometrics, etc. It will return an *opaque credential* to the Access Control Service. The implementation can take into account customer UI styles.

Authentication Service: This service takes the *opaque credential* as input; authenticates it against a standard authentication server, such as Lightweight Directory Access Protocol (LDAP) or a database; creates an *opaque token* if the credential is valid. An implementation of the Authentication Service interface could be X.509 or 3-factor biometrics based on the authentication technology at the deployment site.

Authorization Service: The service takes the *opaque token* as input, fetches the user privileges from a centralized authorization server, such as LDAP server. Like Authentication, this service can be customized for particular technology and business process at the deployment site. In addition to *opaque token*, Authorization Service may provide other interfaces that accept user ID, X.509 certificate as the input of user identity.

The contract between the three services allows their implementations to be hidden and varied. However, it is likely that the three implementations will have to agree upon one or more formats for credentials. For example, in an environment where smartcards that contain user X.509 certificates are deployed, the smartcard implementation of Login Service should be loaded with the X.509 certificate (PKI) implementation of Authentication Service. The data contract between Login Service and Authentication Service is the *opaque credential*; the data contract between Authentication Service and Authorization Service is the *opaque token*. There are many IT standards that can be used to define the syntax of *opaque credential* and *opaque token*. For example, WS-Security defines Username Token Profile, X.509 Token Profile, Kerberos Token Profile and SAML Token Profile.

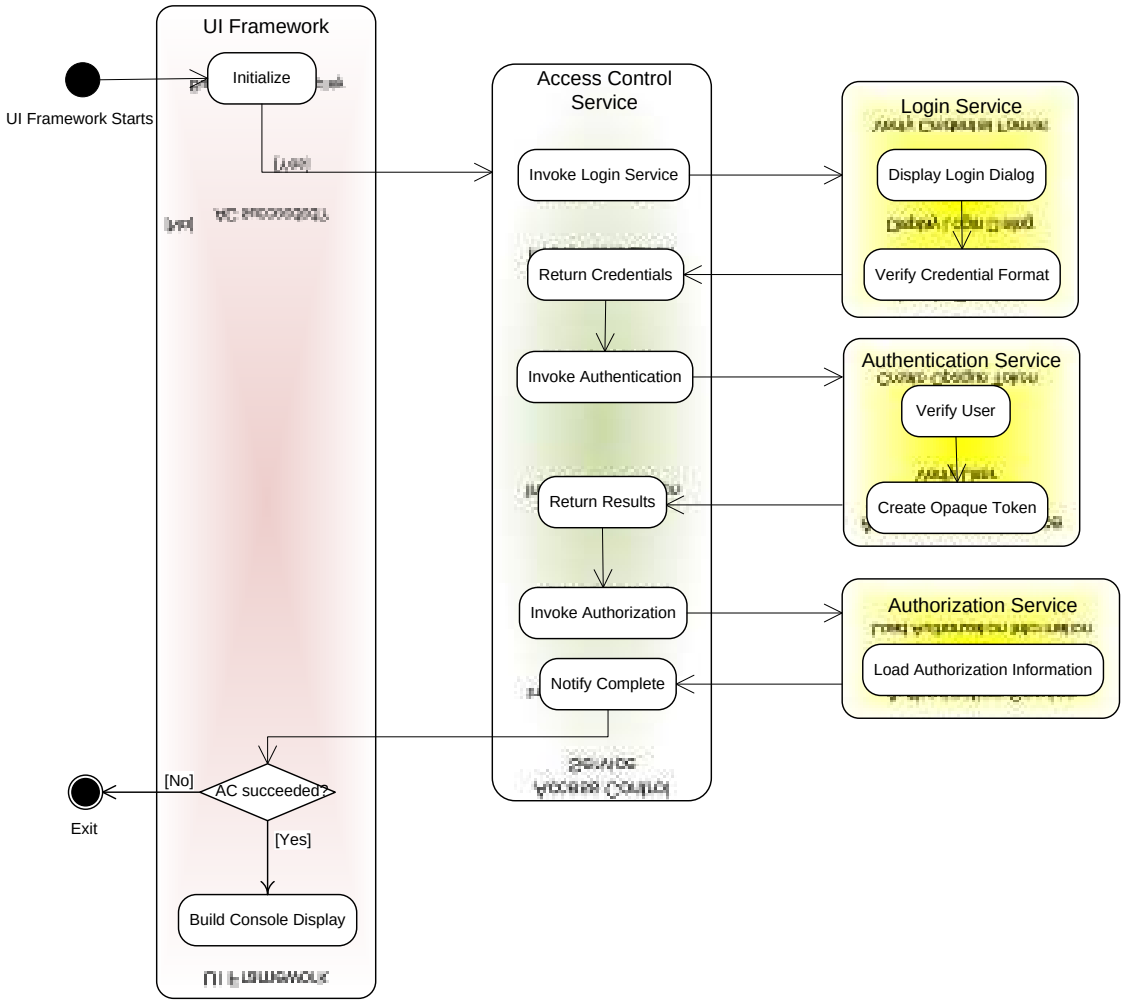

Figure 9-2: Auth/Auth and Login Interaction

9.3.6 Centralized Audit and Monitoring

An audit trail is the data collected from various systems logging activities. It is a chronological record of system activities that enables the reconstruction and examination of the sequence of events and the changes of events. In security perspective, logging and auditing are necessary for identifying policy violations and fraudulent activities; for incident report and response; for forensic analysis; and for government and industry regulation compliance. For successful log correlation, closely synchronized time is required between all network devices. This section addresses the audit and monitoring functionality provided by this security architecture.

9.3.6.1 The Goals

As information security technologies evolve, many typical IT security solutions are used in systems such as firewalls, IDS/IPS, VPNs, antivirus, authentication servers, and patch management systems. These devices and systems have helped to address specific security issues, but they also generate a great deal of logs. Adding those logs to the user activities/events logged by typical EMS applications, is a matter of literally millions of events

each day that should be monitored, analyzed, correlated, and archived. The goals of this security architecture for centralized audit and monitoring are as followings:

- Provide scalable and centralized logging functionality that limits unnecessary network bandwidth consumption and minimizes system performance degrading;
- Provide access control, encryption and data integrity service to secure log entries;
- Support log backup and archiving;
- Support post internationalization (formatting of numbers and date/time);
- Enable log correlation by standard IT Security Information/Event Management (SIM/SEM/SIEM) systems;
- Enable industry regulation compliance to NERC CIP-002-1 through CIP-009-1, SOX etc.;
- Support enterprise solution integration.

9.3.6.2 Functional Architecture - Common Logging Service

In legacy EMS Systems, each application generates log entries that have inconsistent content and format. For example, one application logs user name, but another application logs IP address; some applications use Local Time, others use GMT; some applications store log entries in plaintext files, others use database as log repository. Proprietary logging function in different applications is the main challenge to log management and centralized auditing and monitoring. This architecture uses the Common Logging Service that provides log generation, protection, monitoring, archive and localization for EMS applications based on industrial standards. It can be used as a service in a Service Oriented Architecture (SOA) solution or connect to Integration Layer readily.

Common Logging Service is composed of three interfaces, log/query interface, format interface, and repository interface, as shown in Figure 10.3.

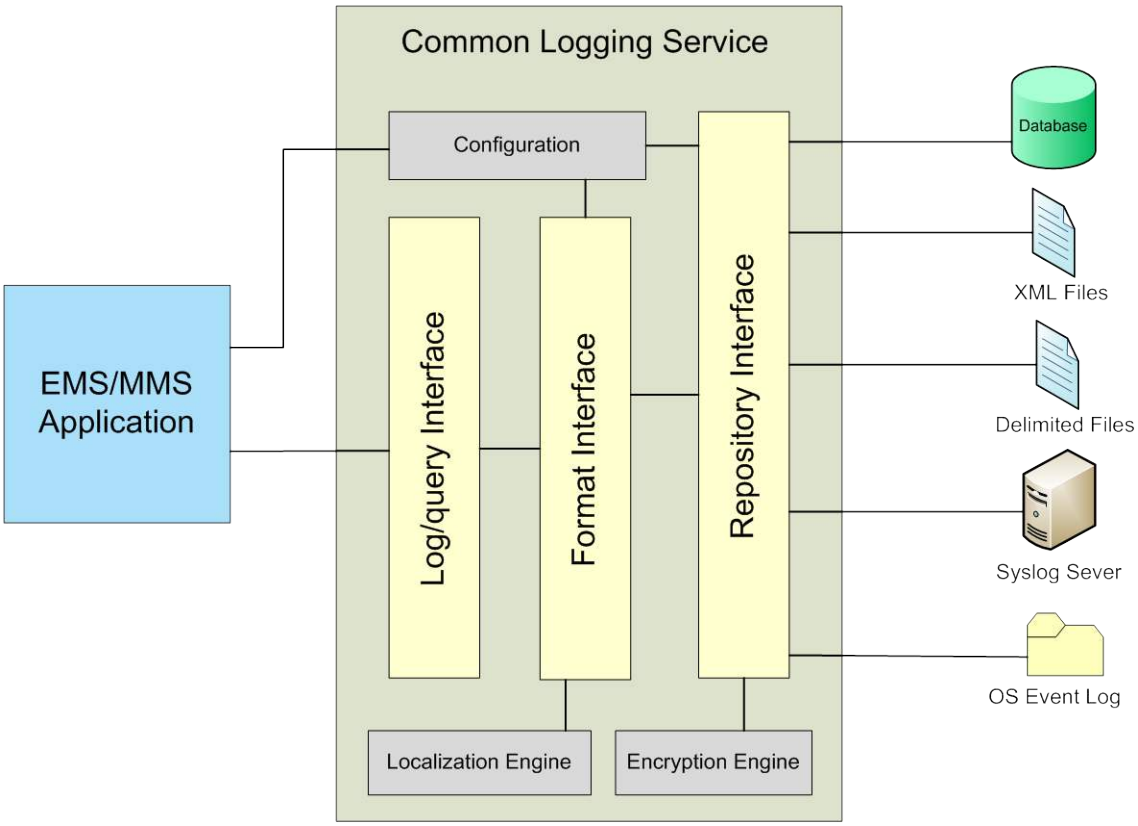

Figure 9-3: Common Logging Service Architecture

Log entry should contain sufficient information for troubleshooting, log correlation, system auditing, forensic analysis and regulation compliance. The following basic information is mandatory.

- *Message type – known as a facility defined by syslog protocol (refer to RFC 3164 – The BSD syslog Protocol). Examples of facilities include kernel messages, authorization messages, and user level messages.*

- *Severity – each message is assigned a severity value 0 (emergency) to 7 (debug) as defined in RFC 3164. The combination of facility and severity value is called the priority value of a message. The message priority value is usually used by SIM/SEM/SIEM solutions to determine which messages should be handled more quickly. However, the priority value does not affect which actions are performed on each message.*

- *Time stamp – Local time should be used for time stamp for consistency and log correlation. The format should be consistent with the standard timestamp format used by different logging standards. For example, use syslog timestamp format if the repository is syslog server; use Windows Event Log format if the log entry is written to Windows Event log (Note: Special considerations may be required for systems that span multiple timezones). All times should be in sync with each other (via ntp v4) for this to be successful.*

- *User ID – User ID should be logged for NERC CIP compliance. The format of user ID depends on the authentication method used in the system, but user ID should be able to uniquely identify an authenticated user who performed or attempted to perform the action.*

- *Workstation/host ID – workstation/host ID is required. It can be IP address, Media Access Control address (MAC), computer name that can identify a workstation or a host uniquely. In a server application, both the workstation ID that issued a command and the host ID that carried the command should be logged if technically feasible.*

- *Description of activity/event – The activity, command, or event should be logged. If the action is changing a critical system configuration, both old and new value should be logged.*

- *Sensitive information, such as password, encryption key, network path and etc., should not be logged.*

The **configuration interface** may provide Application Programming Interfaces (API)s for setting, changing and getting logging settings. The settings include log entry format, default language, log repository, encryption and more. Encryption should not allow logs to become unuasble if a user is removed. Configuration can also be performed in a non-intrusive manner, through configuration files.

The **log/query interface** provides standard APIs for EMS applications to log messages to the configured repository in specified format.

The **format interface** provide standard APIs for formatting log messages to the configured format. Multiple implementations of the Format Interface can be plugged-in to support different log format standards, such as syslog, Windows Event log, XML file, comma delimited log entry and any other future standards.

The **localization engine** provides service of localizing log messages to the specified language for viewing. It is post-internationalization which means a log message is only localized when it is fetched for viewing or printing. It is not localized in the storage. This design allows a log entry being viewed or printed in different languages on demand.

The **repository interface** provides standards APIs for saving/fetching log entry or entries to/from log repository.

Different types of repositories can be supported by supplying specific implementations of the repository interface. Figure 10.3 illustrates several common repositories. The purpose is to show the concept at architecture level, but not to provide an exhaustive list.

A logging storage can be a remote database, or a local plaintext file, or a distributed syslog server. The architecture provides system designer and integrator the flexibility of choosing appropriate technologies for meeting specific deployment requirements, such as host performance and network bandwidth.

The **encryption engine** provides a service for log entry encryption and integrity protection. Data encryption and data integrity should be provided by a logging solution. However, a deployment of a logging solution can leverage the encryption features built-in the repository, such as Oracle

database, XML encryption and Windows EFS, to provide such services. The technology decision should be made based on the holistic view of a system and the enterprise integration strategy.

The same philosophy applies to access control of the logs. Authentication and authorization should be in place for saving and fetching log entries. However, reinventing the wheel for the logging service is not recommended. Standard service-to-service authentication and authorization technologies should be used for the Common Logging Service. See section 9.3.5.2 for authentication and authorization services.

This architecture does not post any restriction on backup, archive, auditing and monitoring mechanisms. Standard log formats and repositories enable the integration with enterprise centralized audit and monitoring solutions, such as SIM, SEM and SIEM. Common Logging Service is a foundation for centralized audit and monitoring. An Energy Management System or Market Management System could choose to implement some log correlation, monitoring and/or alarming features based on business needs. A SIM/SEM/SIEM product could be integrated to automate correlation, consolidation, backup, archive, compliance monitoring, incident report and response; and to provide centralized management with in a system or at the enterprise level.

9.3.7 Change Management and System Integrity

Changes can be instigated from a variety of sources, some are planned, some are by accident or mistake, and some are malicious. Planned changes include deployment of security patches, system and application upgrades, and managed configuration changes. Unplanned changes could be administrators' or operators' mistakes, or hacker activities. Change Management includes the workflow and the approval processes for managing the changes; the deployment of patches, new releases and configuration updates; and change auditing. Automatic Change Auditing is another layer of security defense that is able to detect and stop malicious changes to the system. It ensures system integrity and maintains the security posture and continuous compliance.

The objective of this section is to address the ability of automatic auditing of changes and configurations in EMS Systems for system integrity and compliance. The specific goals are:

- Simplify patch management and upgrade processes;
- Detect changes automatically to ensure timely incident response and improved availability of mission-critical systems;
- Enforce policies and regulation compliance;
- Be consistent with corporate/enterprise IT management solutions to maximize enterprise integration for reducing operational cost.

This section first discusses Automatic Change Auditing then describes two reference implementations in the context of this security architecture.

9.3.7.1 Automatic Change Auditing

The typical change process involves

- planning the change
- approving the change
- performing factory acceptance testing
- performing site acceptance testing,
- deploying to production
- maintenance

The process can be streamlined and even automated using the latest technologies. Automatic change auditing works parallel to the change process for detecting inappropriate changes to the system in a timely manner so that mis-configuration can be corrected and cyber attack can be mitigated promptly. Automatic change auditing as supported by this security architecture uses change and configuration management technology. A trusted baseline configuration snapshot tested in the factory acceptance test is used as the measurement standard. Automatic change auditing detects and reports all changes to the trusted baseline. The following additional features are supported.

- An **automatic discovery** feature that detects new devices and/or new applications in a system. With hundreds of systems, devices and software installed in a system, it is difficult to keep track of all installed hardware and software. Many common cyber attacks install RootKit and other malicious agents as the "stepping stone" of penetrating and compromising a system. An automatic discovery feature helps administrator find unwanted devices and software in time for incident response and system management.

- Allow **modeling of specific security policies** and/or regulations for compliance. Different utilities may have different security policies and need to conform to different set of regulations. The ability of modeling specific policies and regulations is essential to the architecture. Any change that conflicts with the modeled policies and/or regulations is detected and reported.

- Provide an **independent audit trail** so that every change is recorded and detailed audit trails are established for incident response and forensic analysis. It is important that the audit trail of Automatic Change Auditing is independent of the audit trail provided by automatic deployment tools used for deploying patches or system upgrades.

- Provide **multiple levels of reporting and alerting** serves different audiences, such as EMS administrator, IT executives, and compliance auditors.

- Allow for **rollback of inappropriate changes** to a pre-existing state. Even though a change has gone through FAT and SAT, administrator may find that the change degrades a system in real-time and is not acceptable in production, "rolling back" the system to a pre-existing state is critical for resuming normal business operations.

- Integrate with an enterprise change and configuration management system This can allow the corporate security officer, CIO and compliance officer to pull out a global view of the audit report at the enterprise level or at the business function level.

9.3.8 Shared Disk Storage

Shared disk storage is a popular enabling technology strategy for decreasing costs and increasing the return on investment. Often overlooked from a security view, shared disk storage present special concerns and issues that need to be addressed when implementing any system, especially when storage resources are shared between assets across multiple security domains.

The two most popular network and infrastructure configurations (NAS and SAN) are discussed below. Each has its own unique set of security issues that should be addressed when deploying them as storage solutions. Please note that the illustrations presented are showing interaction between Level 4 and Level 2 assets. The concepts presented are applicable to other levels.

9.3.8.1 Network Attached Storage (NAS) Shared Storage

NAS shared storage arrays (also called appliance or filers) are inherently insecure when used for the sharing of data across multiple security domains. For any NAS network, there are challenges and issues that need to be overcome:

- Filer OS (most modern NAS use a proprietary Linux kernel optimized for performance and not necessarily for security. Firewall rules can be hardened but the big issue is logging and auditing)

- IP Security (SSH or some other strong encryption scheme should be used. In many cases the financial and performance costs associated with the NAS host providing strong authentication is prohibitive.)

- IP Infrastructure (whether using NFS or iSCSI, security hardening is mandatory for both client and server. In some cases applications may have to be modified. LANs should never be allowed to cross a security perimeter without PEPs in place for each port connection. If the NAS filer cannot provide PEP functionality, the burden falls on either the client/server or the switch (or both) to provide the mandated authentication and logging.)

Recent advancements in technology have definitely helped the situation, but questions still remain, especially those centered on logging and auditing abilities.

A rule to live by is to have one NAS farm (group of appliances) for each security zone. Stated differently, NAS appliances can be used locally within the Level 4 or Level 2 zones, but the local LANs to which they are connected should not extend beyond that zone's security perimeter. Following common sense security guidelines still apply.

NFS or iSCSI over IP can still be used to route traffic destined for shared storage across security perimeters under certain conditions (see NFS and File Sharing below).

9.3.8.2 Storage Attached Network (SAN) Shared Storage

Using SAN for shared storage offers advantages in overcoming some of the security risks inherent with NAS because of the technology used: implementation of fiber fabrics and the use of fiber channel protocols. However, to fully meet security requirements, certain precautions should be applied and followed:

- The placement of SAN storage appliances should always be within the perimeter of the highest level security domain from which access will be controlled, in this case Level 4 or the Blue Zone. All access from the Blue zone, as well as other zone servers and devices (namely Level 2 or Yellow Zone) will be tightly controlled as defined below.

- A successful fiber fabric connected storage deployment across multiple security domains is dependent on combining connecting technologies and strategies, namely using vSAN (virtual SAN), and implementing a technique called zoning (not to be confused with Security zones and domains as described elsewhere in this document). The key fact behind vSAN and zoning is that each asset can only see what is purposely defined within a highly restricted security enhanced environment

Virtual Storage Attached Network (vSAN)

A vSAN is a collection of ports from a set of connected Fibre Channel switches that form a virtual fabric. Ports within a single switch can be partitioned into multiple vSANs despite sharing hardware resources. Conversely, multiple switches can join a number of ports to form a single vSAN. In addition to offering different high-level protocols such as FCP, FCIP, FICON, and iSCSI, a vSAN is a separate self-contained fabric using distinctive security policies, zones, events, memberships, and name services. Traffic is also separate. One advantage of a vSAN over a typical fabric is that it can be resized port-by-port rather than switch-by-switch.

9.3.8.2.1 Zoning

To further refine the security requirements for shared storage, vSANs allow for zoning to be deployed in the carving out or isolating different and electronically separate groups of disk partitions.

Zoning is the partitioning of a Fibre Channel fabric into smaller subsets to restrict interference, add security, and to simplify management. If a SAN contain several storage devices, each system connected to the SAN should not be allowed to interact with all of them. Zoning is also different from vSANs, in that each port can be a member of multiple zones, but only one vSAN.

There are two main methods of zoning, hard and soft, that combine with two sets of attributes, name and port. Soft zoning restricts only the fabric name services, to show the device only an allowed subset of devices. Therefore, when a server looks at the content of the fabric, it will only see the devices it is allowed to see. However, any server can still attempt to contact any device on the network by address. In this way, soft zoning is similar to the computing concept of security through obscurity. In contrast, hard zoning restricts actual communication across a fabric. This requires efficient hardware implementation (frame filtering) in the fabric switches, but is much more secure.

Zoning can also be applied to either switch ports or end-station name. Port zoning restricts ports from talking to unauthorized ports. Because this is non-standard, it usually requires a homogeneous SAN (all switches from one vendor). Any device plugged in a specific physical switch port is given access to the zone. Name zoning restricts access by device's World Wide Name. This is more flexible, but WWNs can be spoofed, reducing security.

A common sense approach is to use a combination of hard and name zoning methods to create major groups of disk partitions.

9.3.8.2.2 NFS and NAS File Sharing across Security Domains

In most cases, a need exists to provide NFS shares in order to allow the sharing of common files and data between Level Two (Blue Zone) and Level Four (Yellow Zone) components (i.e. between market systems and EMS).

The key solution is to have a shared SAN disk volume where both critical and non-critical assets have access. The fiber interface to this partition uses the same approach defined above (vSAN and zoning), but uses an interface appliance provided for NFS sourced files. One side of the appliance is fiber attached to the SAN while the other side is connected to the IP LAN containing NFS clients and servers. All NFS data passing through this appliance is translated to FC protocol and visa-versa.

Level Four (Blue Zone) critical assets have full read/write access to the shared volumes, while Level Two (Yellow Zone) assets should use strong authentication to read or write to this area.

Most SAN storage vendors offer NAS to FC appliances that provide the necessary PEP features.

This appliance is considered a critical asset and should reside within the Level Four (Blue Zone) security perimeters.

9.3.8.2.3 Securing Direct Access to Shared SAN Storage

Because a SAN appliance is considered a critical asset within Level Four (Blue Zone) as per the definition above, strong authentication and extra precautions should be used for any direct access originating from any one of the four other zones.

Most SAN appliances have a local console, sometimes called a Control Center Console (CCC), used for local maintenance of the SAN appliance and other related appliances (such as the NAS to FC PEP translator described above). A CCC is considered a critical asset and should reside within the Level Four (Blue Zone) security perimeters. Because it can be accessed from other zones, strong security and authentication should be in place and PEP functionality should be provided.

Most SAN appliances also provide a gateway to allow vendor remote diagnostic access and to provide a calling mechanism as a way for the local system to notify the vendor in the event of a failure. Dual firewalls should be used to provide protection for access which is also subject to strong authentication (for example VPN tunnel and/or RSA Token). An additional certificate key should be used by the vendor.

9.3.9 Server Virtualization

Like shared disk storage, server virtualization is an enabling technology used for reducing costs and resources. Virtualization as a technology has matured from it was just a few years ago, with new products providing architectural enduring qualities such as availability, assurance, usability, and adaptability. This section’s focus is on the security aspects within the architectural context of this document and no attempt is made to present views on what can or cannot be virtualization. To begin the discussion on security, an overview of the technology is in order.

9.3.9.1 Server Virtualization Defined

Server Virtualization (also called platform virtualization) technology is based on the concept of system virtual machines. A virtual machine is capable of virtualizing a full set of hardware resources, including a processor (or processors), memory and storage resources and peripheral devices. A virtual machine monitor (VMM) is the piece of software that provides the abstraction of a virtual machine. There are three properties of interest when analyzing the environment created by a VMM:

- Equivalence - A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
- Resource control - The VMM should be in complete control of the virtualized resources.
- Efficiency - A statistically dominant fraction of machine instructions should be executed without VMM intervention.

Most virtualization projects and offered products are focused on x86 architectures, although most major vendors such IBM, HP, and Sun offer services and products directed to their respective technology base. This section focuses on the x86 platforms but the concepts can apply to any implementation and platform.

The basic configuration of a virtualized environment consists of a physical host server (or servers in a high availability cluster configuration) running a proprietary OS and containing physical CPUs, memory, disk storage, and peripherals as illustrated below [9].

---

[9] The above illustration was adapted from the NERC CIP-Compliant Architecture Pattern recommendations to the US ISO/RTO IT Committee (ITC)

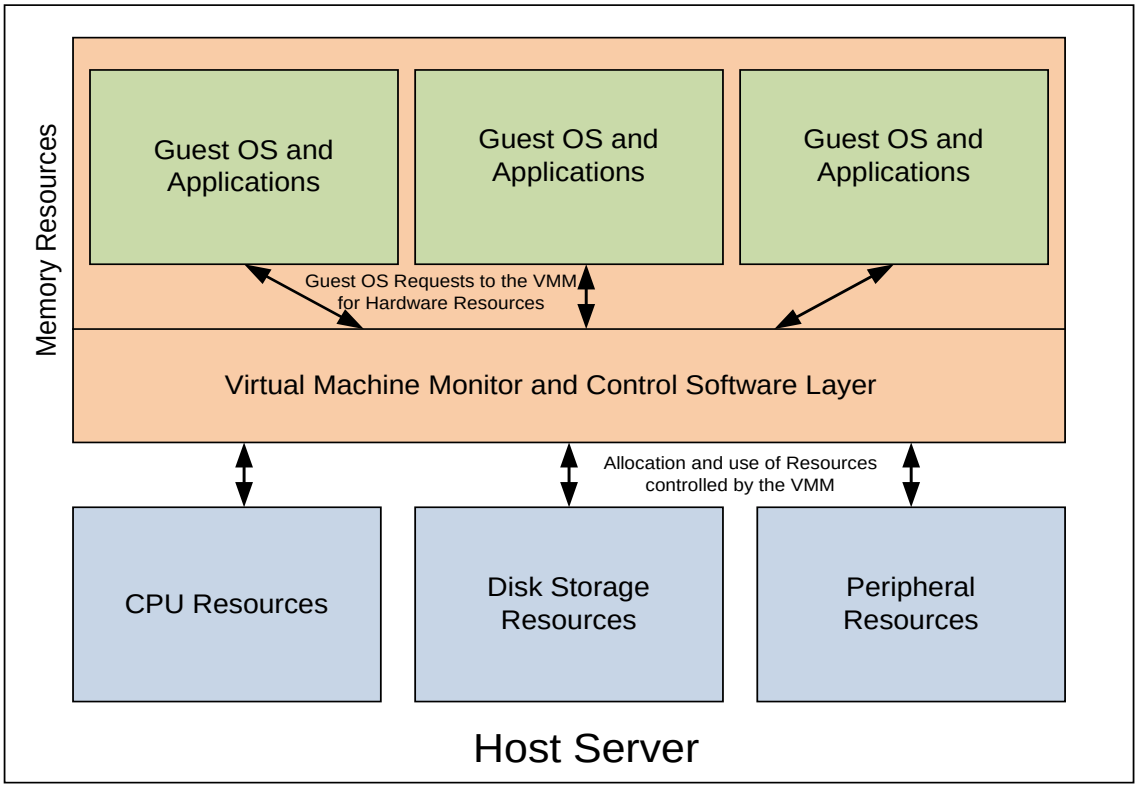

Figure 9-4: *Typical VM Environment*

Controlled by a VMM software layer, memory is segmented (or partitioned) in which contain images of individual “guest” OS’s and their applications. Executing images that make requests for hardware is intercepted and managed by the VMM. Guest attributes such as CPU usage and priority are controllable during configuration time, or can be automated to meet maximum/minimum performance criteria.

9.3.9.2 Security Issues

Security hardening of each guest image can be done almost as easily and effectively as if the image was running on its own dedicated server following the same common sense security practices and mandated standards. Like SAN storage solutions, virtual clusters and farms are usually controlled by a central console and the same type of security precautions should be observed. However, unlike the SAN storage solution which takes advantage of certain technologies, i.e. vSAN and zoning, special consideration should be given to virtualized systems when discussing security zone perimeters.

When designing large scale data processing systems, the creation of large virtual clusters or virtual server farms to realize large cost/benefit ratios is desirable. But be aware of the issues that will arise when guest images from different security zones are mixed on the same virtual system as illustrated below. [10]

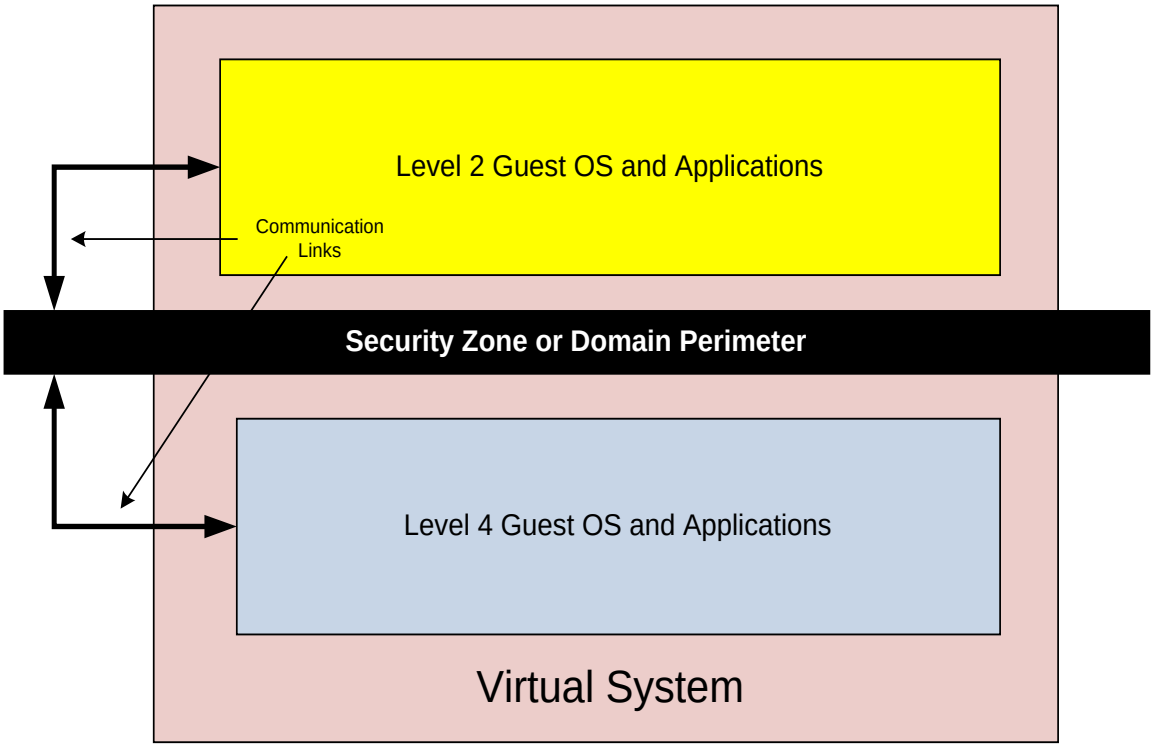

Figure 9-5: *Shared ESP and non-ESP on Common VM*

The most pervasive security issue facing virtual systems is that shared memory constitutes, in most cases, an unacceptable vulnerability for attacking one guest image from another that is supposedly isolated by PEP devices and controls. A virtual system should have the necessary control mechanisms in place to adequately isolate the memory space of guest images from each other and isolate network communications between Level 4 applications and other lower level assets, insuring that communications is truly controlled by the perimeter PEP devices.

Other issues:

---

[10] The above illustration was adapted from the NERC CIP-Compliant Architecture Pattern recommendations to the US ISO/RTO IT Committee (ITC)

- The likelihood that component failures on a level 4-x process could directly affect the functioning of Level 4 applications (some vendors are now providing high availability and redundancy features in their cluster technology to negate these effects).

- There is difficulty in determining actual vulnerabilities within a proprietary system design which limits the level of trust possible within a specific virtual system vendor's offering.

- Although logical safeguards could be implemented in software, there still exists the fact that hardware resources are shared through a common backplane, presenting, although low level, a persistent security risk. Very few vendors provide logical separation via the hardware, relying mostly on the mostly on software solutions.

9.3.9.3 Separation of Virtual Systems

The issues presented above can be corrected by physically separating virtual systems, clusters, and farms based on the category of applications running under the guest images as shown in the example figure below. [11] Stated differently, each virtual system that does not have adequate built-in control should contain only guest images belonging to the same security zone.

---

[11] The below illustration was adapted from the NERC CIP-Compliant Architecture Pattern Recommendations to the US ISO/RTO IT Committee (ITC)

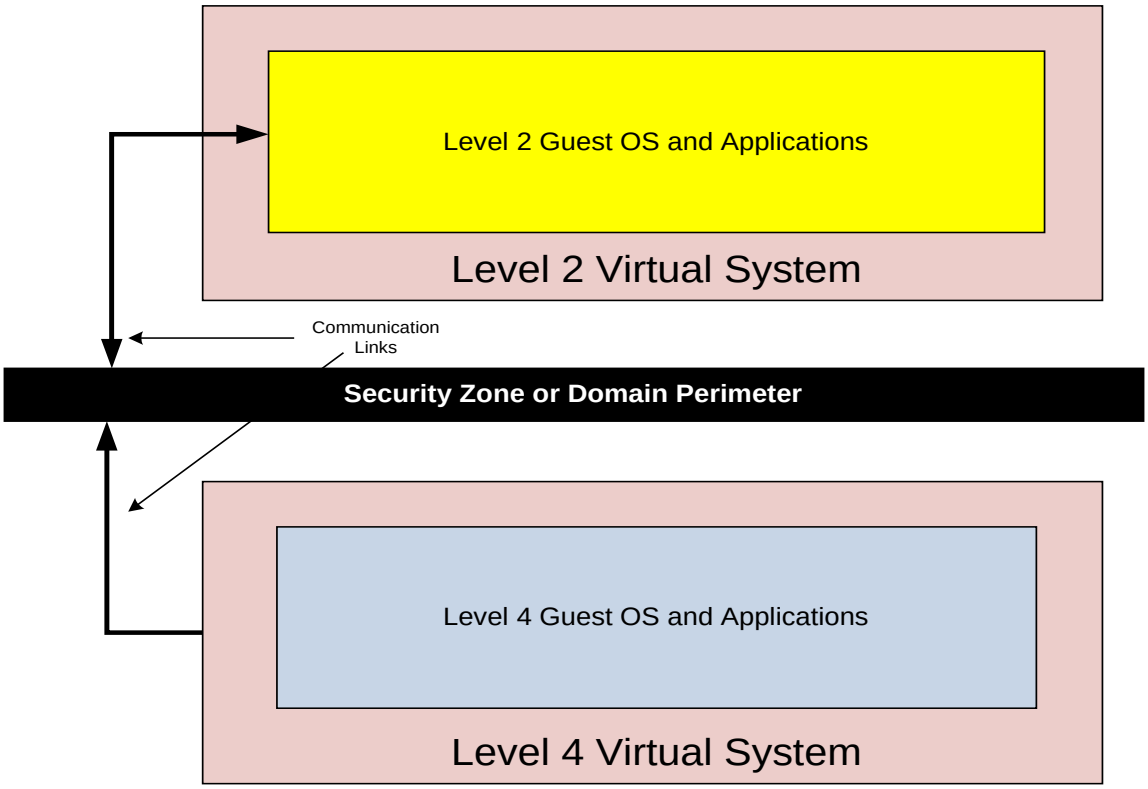

Figure 9-6: *Separate ESP and non-ESP on Separate VMs*

There are other advantages (and one disadvantage) of separating virtual systems. The one disadvantage is the increase in capital cost and maintenance as this approach requires an increase of physical, software, and network infrastructure. The other advantages include:

- A decrease in the number of guest images and associated applications that would be subjugated to audit and strict compliance to security regulations such as NERC CIP or the international equivalence
- Increase in confidence (trust) in meeting compliance standards
- Systems can administered independently when applying firmware updates, application patches, hardware replacement, and other changes

9.3.10 Conclusion

The cyber security and network architectures provides a framework that supports all of the energy management and market management business requirements and business processes in an open but secured network environment. It adheres to key security design principles. It focuses on building layered security defenses at different levels (network, system, application); and providing common security elements (authentication, authorization, auditing, administration)

at all levels. The fundamental goal of applying these principles and design elements is to provide a security infrastructure that meets business needs in three core areas: availability, integrity and confidentiality. Security is only feasible with a systematic approach which is applied across all levels throughout the life cycle of an EMS system. Security processes required include defining security roles, establishing security policies, identifying critical assets, vulnerability and threat assessment, risk evaluation and management, incident report and response process, change management, and disaster recovery planning. This security architecture combined with security processes and robust applications ensures a reliable and resilient system.

9.4 Applicable Standards

- NIST Special Publication 800-27 Rev A - Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A
- Roadmap to Secure Control Systems in the Energy Sector, DOE http://www.oe.energy.gov/DocumentsandMedia/roadmap.pdf
- NIST Special Publication 800-53 Recommended Security Controls for Federal Information Systems
- NIST Special Publication 800-82 Guide to Industrial Control Systems (ICS) Security
- MS-ISAC - Cyber Security Procurement Language for Control Systems Version 1.6
- INL - Mitigations for Security Vulnerabilities Found in Control System Networks - http://www.inl.gov/scada/publications/d/mitigations_for_vulnerabilities_in_cs_networks.pdf
- OWASP Project: http://www.owasp.org/index.php/Category:Principle
- Secure Architecture Design, http://csrp.inl.gov/Secure_Architecture_Design.html
- NISCC Good Practice Guide on Firewall Deployment for SCADA and Process Control Networks
- Method for Architecting Secure Solutions – IBM Enterprise Security Architecture Redbook. http://www.redbooks.ibm.com/redbooks/pdfs/sg246014.pdf
- RFC 3164 – The BSD Syslog Protocol, http://www.faqs.org/rfcs/rfc3164.html
- Configuration Audit and Control: 10 Critical Factors to CCM Success, http://www.tripwire.com/resources/asset_request.cfm?aid=9796
- Gartner Analyst Report: "CMDB or Configuration Database: Know the Difference"
- IEC 60870-4 (1990-04) - Telecontrol equipment and systems. Part 4: Performance requirements

- IEC 60870-5-104 (2006-06) - Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles

- IEC/TS 60870-5-604 (2007-10) - Telecontrol equipment and systems - Part 5-604: Conformance test cases for the IEC 60870-5-104 companion standard

- IEC/TS 60870-6-503 (2002) - Telecontrol equipment and systems - Part 6-503: Telecontrol protocols compatible with ISO standards and ITU-T recommendations – Services and protocols

- IEC 60870-6-702 (1998-10) - Telecontrol equipment and systems - Part 6-702: Telecontrol protocols compatible with ISO standards and ITU-T recommendations - Functional profile for providing the TASE.2 application service in end systems

- IEC 60870-6-802 (2005-09) Ed. 2.1 - Telecontrol equipment and systems - Part 6-802: Telecontrol protocols compatible with ISO standards and ITU-T recommendations - TASE.2 Object models

- EC 61850-90-2 (Future / 2010) - Communication networks and systems for power utility automation - Part 90-1: Communication between substations and control centers using ISO / IEC 61850

- IEC TS 61850-80-1 (Future / 2008) - Communication networks and systems for power utility automation - Part 80-1: Guideline to exchange information from a CDC based data model using IEC 60870-5-101/1 04

- IEC 62351 Data and Communications Security

    - 62351-1: Introduction

    - 62351-2: Glossary

    - 62351-3: security for profiles including TCP/IP

    - 62351-4: security for profiles including MMS (ISO 9506)

    - 62351-5: security for ISO/IEC 60870-5 and derivatives (DNP)

    - 62351-6: security for ISO/IEC 61850

    - 62351-7: objects for network management (future)

10. Platform and Business Continuity Architecture

10.1 Overview

The key objectives of the Platform and Business Continuity (BC) Architecture are to define an open computing platform to support application systems and providing the following capabilities.

- Multiple system instances including test, training, and production application systems
- Continuous operations in spite of local anomalies and planned maintenance
- Disaster recovery in case of major system failures or local disasters

The computing platform includes those hardware, system software, and infrastructure components that are generally provided by vendors that specialize in providing hardware, operating systems, database management software and middleware products. An application vendor would utilize these computing components as the computing platform that supports the application.

10.1.1 Computing Components

The platform computing components include the following:

- Hardware components:
    - Application Servers
    - Database Servers
    - Storage Devices
    - Network Devices
    - User Interface Consoles
- Software components:
    - Operating Systems
    - Database Management Systems
    - Middleware Software and Integration Software
    - Networking Software
    - Infrastructure components
        - Storage networks

- Storage snapshot features
- Backup features
- Clustering features
- Virtualization features

10.1.2 Computing Platform System Images

The computing platform should have the flexibility to support test, training and production systems. Further, the computing platform should be capable of supporting multiple redundant production systems either locally or remotely to support local and remote control centers. The requirements for the system hardware and software components of the computing platform are intended to support continuous operations in spite of local disruptions or disaster scenarios as well as supporting system changes or upgrades with minimal to no impact on operations. This requires that data sets and displays be kept in sync across the computing platforms.

To support business continuity the computing platform provides for multiple production system images that act as a backup for other system images on a local basis and across remote data centers or control centers. This redundancy extends down to the independent platform components of the application including the application servers, database server, and storage system.

10.2 Scope

10.2.1 Computing Platform Requirements

The general requirement is that application systems are supported on open, standard computing platforms composed of hardware and software components that are generally available in the market. In addition, the application systems will have the flexibility to support the various computing components as described above from different major vendors with defined roadmaps for future system direction and enhancements. It is expected that the application system will utilize features of the computing platform to support requirements such as system reliability, scalability, flexibility, availability and performance.

Long-term the computing platform will provide multiple servers for computing and storage. The application would not be tied to specific computing assets or rely on any specific server or set of servers. Rather, the application would submit a request for computing resources and the computing infrastructure would choose the best path and resources available to serve the request.

Of particular interest is the objective of vendor database management system independence. The platform solution should support multiple leading database management software products based on open SQL standards.

10.2.2 Business Continuity and Disaster Recovery Requirements

To support business continuity and disaster recovery requirements the platform architecture will be designed to provide redundancy to support continuous operations while providing flexibility to react to component failures or disaster scenarios. The architecture will also support maintenance or system upgrades with no or minimal impact on operations. Consequently, the following options should be supported:

- Modes of Operations – The architecture will have the capability to support various modes of operations between the production systems including:
    - Peer to Peer mode – Dual systems which are both primary. Each system may be redundant or non-redundant
    - Split mode – Segregation of systems particularly for support of maintenance and upgrades
    - Primary/Secondary mode – One system acts as primary and is synchronized to the secondary system.
- Synchronization – The architecture will utilize industry standards to provide synchronization of function and services between production systems and platform components within a specified transfer time for each application or business service.
- Storage – The architecture will allow for the use of storage hardware based on snapshots and data replication as a means for performing backups and fast recovery.
- Flexibility - The architecture will provide the capability to configure applications and business service components such that modes of operations and synchronization requirements can meet the specific business requirements.
- System Maintenance and Upgrades - The architecture will support the upgrade or maintenance of application, system software and hardware components of one system image independent of other system images. In addition, a single model maintenance master tool will be employed to synchronize and control the propagation of model updates between system images across all computing platforms
- Monitoring and Control – A master console will be employed to monitor system components in the architecture, provide control over modes of operation, and support system upgrades and maintenance.
- Service Provider Conflict Resolution - Services in a high-availability environment should automatically resolve competing service providers.

10.3 Architecture Description

The figure below provides a high level, abstract view of the major components of the business continuity architecture.

The objective of this architecture is to provide operational options for redundancy and fail-over within a data center and also across remote data centers.

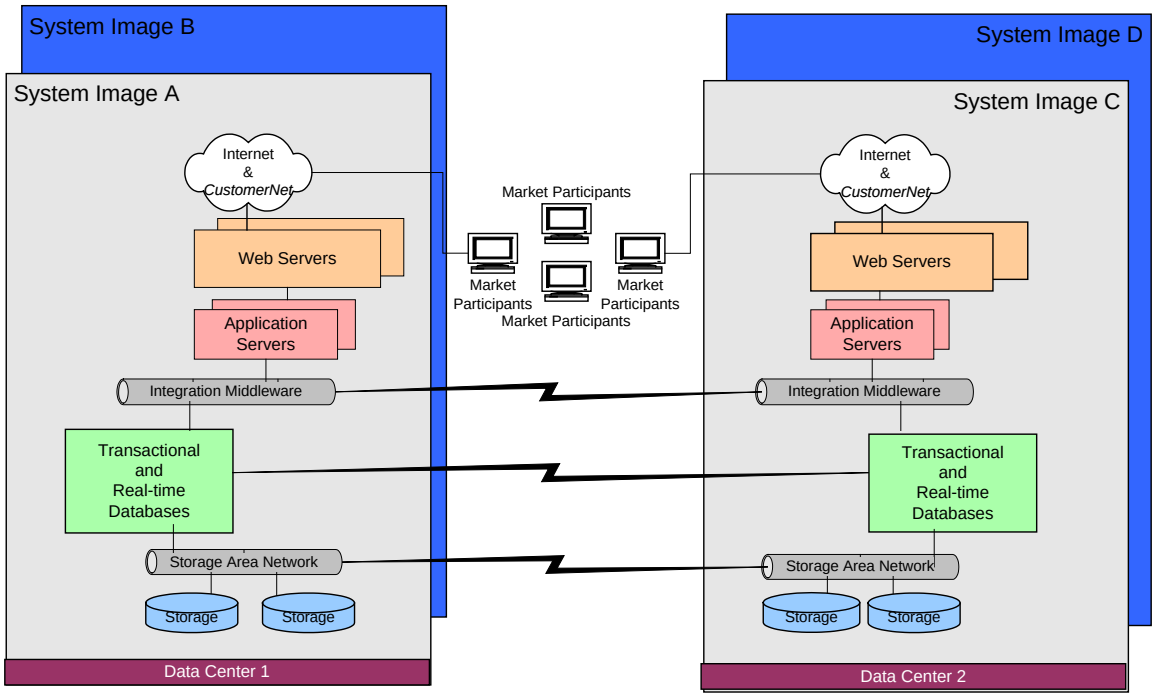

Figure 10-1*: Platform and Business Continuity Logical Architecture*

10.3.1 Data Synchronization

The architecture will ensure that data, including operational data such as alarms, operator tags, and manual entries, are replicated and synchronized between production system images. In peer-to-peer mode when two systems are operating in parallel the system will manage which system or system component represents the system of record and to support data replication and synchronization. The capability to control and view the status of the system or record should also be provided. Generally available third-party vendor tools should be utilized where available. To provide flexibility and redundancy for critical real-time data inputs, the architecture will support the acceptance of real-time data such as SCADA, ICCP or meter readings by two systems simultaneously.

10.3.2 System Maintenance and Upgrades

The architecture will support the outage of platform components to support planned or unplanned maintenance or upgrades by isolating a system image or component from the operational system of record. Platform components include server, database, and storage

devices. Of particular interest is the maintenance or upgrade of model data such that model updates can be coordinated and synchronized across system images.

10.4 Applicable Standards

- SQL standard - ISO/IEC 9075:1992
- NERC EOP-008

11. Glossary

Term	Definition
Abstract Component	A method of categorizing an application, group of application, or a piece of an application which provides standardized services that can be invoked (or consumed) by various independent applications.
Canonical Data Model (CDM)	A specification of the information model from which Data Sets are defined.
Common Information Model (CIM)	IEC standard common information model that is intended for use as a major component of the CDM.
Consumer	A business component that is considered the destination of the Data Set
Data Access Layer	The infrastructure that manages Data Sets, eliminating any direct communication between producer and consumer components.
Data Set	A logical set of business data, defined by publicly available metadata. A Data Set Instance is produced by a source, and made available to other components via services.
Data View	The CDM sub-model that the Data Set creates for consumers. A Data View relies on elements of one or more Messages (whereas a Data Set specification divides CDM according to producer output).
Event / Data Set Event	All or any part of an incremental update of a Data Set that is made available as a notification to subscribers
Incremental update	A producer updates a Data Set instance incrementally if it only issues the differences from the previous version.
Producer	A business component that is considering the source of the Data Set
Message Exchange Pattern	The model used to describe information flows, such as or Broadcast, Request/Reply

Term	Definition
Service Specification	A specification comprised of both a Data Set specification (derived from the CDM) and an exchange pattern (based on the Information & Integration Architectures)
SLA (Service Level Agreement)	A part of a service contract where the level of service is formally defined. In practice, the term SLA is sometimes used to refer to the contracted delivery time (of the service) or performance.
SOA (Service-Oriented Architecture)	A flexible set of design principles to provide a loosely-integrated suite of services that can be used by multiple applications based on a well understood, well defined interface to access them. XML is commonly used for interfacing with SOA services, though this is not required.
TCO (Total Cost of Ownership)	A full cost accounting estimate to include all direct and indirect costs of a product or system over the life of the asset.