

Design and Evaluation of Plug-and-Play Enabled IVHM Architecture

Tarapong Sreenuch and Ian Jennions
IVHM Centre, Cranfield University

ABSTRACT

For Integrated Vehicle Health Management (IVHM) technology to fully achieve its promise, there is a need for integration architecture to support interoperability between multiple vendors' IVHM components and insertion of new IVHM capabilities. To date there have been very limited studies on different approaches in integrating IVHM components. This paper presents design candidates for plug-and-play (PnP) IVHM architecture. The open standard based designs are derived from desired IVHM data flow characteristics and system configuration requirements. The designs and enabling middleware technologies are qualitatively and empirically evaluated for their adequacy and effectiveness. The qualitative assessment focuses on the implementation and system configuration based on different test scenarios. The empirical performance is measured in terms of latency (in both normal and intermittent network connections) and throughput. The results identify the advantages and disadvantages in the different designs of PnP IVHM architecture in which ease of configuration, performance and applicable data buses are the main conflicting objectives.

CITATION: Sreenuch, T. and Jennions, I., "Design and Evaluation of Plug-and-Play Enabled IVHM Architecture," *SAE Int. J. Aerosp.* 8(1):2015, doi:10.4271/2015-01-9001.

INTRODUCTION

Integrated Vehicle Health Management (IVHM) is a major component in new future fleet management paradigm where a conscious effort is made to shift aircraft maintenance from an unscheduled, reactive approach at the time of failure to a more proactive and predictive approach. It involves data processing which consists of capturing data related to aircraft components, monitoring parameters, assessing current or future health conditions and providing recommended maintenance actions. Multiple sensors, processing units and software are required to perform the data processing functionalities. It is to be expected that these components will be supplied from different vendors. A consensus in aerospace industry is that for IVHM technology to fully achieve its promise there is a need in establishing open IVHM architecture [1]. The architecture is essentially the backbone of IVHM systems which facilitates integration and interaction between multiple IVHM components.

There is a significant number IVHM research projects in Europe and US. These individual works are often aimed at a particular IVHM component or problem. It is not clear how a particular newly developed bit of IVHM functionality can be integrated into aircraft systems. Moreover, there are very few studies on IVHM architecture reported in the literature. Most of IVHM architecture related works are on Open System Architecture for Condition Based Maintenance (OSA-CBM) standard. [2, 3, 4, 5, 6] propose an open standard for development and integration of CBM applications which is then further developed to be the current OSA-CBM standard. OSA-CBM focuses on the IVHM functionalities and associated data models. Most OSA-CBM works are notably from Boeing, GE Aviation and

ARL, Penn State University. [7, 8, 9] discuss the benefit of OSA-CBM and report its use in the Boeing and GE research programs. However, due to proprietary nature of the works, [7, 8, 9] fall short in providing details (at the level required to be able to reproduce the work or further the research) on how OSA-CBM can be implemented or how IVHM architecture can be designed to support OSA-CBM. [3, 4, 5] (early works from Penn State during the development of OSA-CBM) demonstrate the proof-of-concept of how OSA-CBM data model can be mapped to XML and DCOM schema and used in a simple client/server IVHM application.

[10,11] describe the conceptual physical architecture of IVHM systems. The papers suggest a distributed and hierarchical data connection between IVHM components, i.e. multiple sensors to a subsystem and multiple subsystems to a vehicle level. The papers also discuss possible options of how IVHM can be integrated into an aircraft avionics network; 1. Dedicated Architecture, 2. Shared Architecture where avionics components are built in with IVHM capabilities and 3. Hybrid Architecture where existing legacy avionics resources are exploited to support modest IVHM capabilities. [1] a NASA publication describes the functional architecture and how it is mapped to on-board and off-board systems. It also discusses the IVHM functionalities in the contexts of required system certification.

It can be seen that until now the literature related to IVHM architecture are mostly on the conceptual and functional levels. There are no detail technical studies on the design and implementation of IVHM architecture which can feedback to the standard bodies, e.g. MIMOSA or SAE. This is probably due to conflicts of commercial interest thereby hindering the development process of consensus.

IVHM architecture. There are still unexplored significant research areas that will benefit towards the development of open IVHM architecture. One often misleading proposal is to develop small scale OSA-CBM demonstrations (with one or two processing units) to proof adequacy of the architecture, e.g. for gearbox, fuel system, etc. These prototypes are unlikely to address the issues related to IVHM architecture, but instead they are more towards the algorithms integration and completeness of data model. In our opinion, IVHM architecture research should also focus on more generic application-independent architecture questions. Those technical areas are: 1) different designs and trade-offs of the underlying distributed data mechanism, 2) scalability of IVHM architecture in particular at the scale of practical aircraft IVHM systems, and 3) quality-of-service (QoS) of IVHM data events. The contribution of this paper is to partially fill the research gaps related to IVHM architecture. The focus of this paper is on the design and analysis (based on experimental evidences) of distributed data mechanism in the contexts of IVHM architecture.

The outline of this paper is as follows: 'Systems Integration' section describes architecture requirements related to IVHM technology insertion and an open systems approach to systems integration. 'Motivation and Design Goals' section describes IVHM data process characteristics and design motivations. 'Distributed Data Mechanism' section outlines different design candidates for implementation of plug-and-play IVHM architecture. 'Middleware and Practical Validity' section summarizes enabling middleware technologies and their practical validity. Qualitative and empirical evaluations of the designed architecture are discussed in sections 'Qualitative Assessment' and 'Empirical Performance'. 'Concluding Remarks' section summarizes trade-offs between the designs and concludes the paper.

SYSTEMS INTEGRATION

Insertion of IVHM Capabilities

A comprehensive IVHM system will be composed of multiple sensors, processing units (both wired and wireless) and software products. These components will be supplied from multiple vendors, and they must be able to operate coherently to provide required IVHM functionalities. Implementing an IVHM system requires significant up-front resources from a maintenance organization. Clearly, sufficient resources may not be available initially to permit near-term application of IVHM across the entire scope of platforms. This means IVHM proponents is likely to initially focus on components with an anticipated high payback in improved performance, more efficiently maintenance capability and overall reduction of platform life-cycle-cost. Hence, the implementation will require being incremental based on necessity (see figure 1). The IVHM system is constantly upgraded at any time when new resources or new IVHM capabilities become available. The IVHM implementation can also include proof-of-concept and prototyping activities that can be applied incrementally, not waiting for a single solution package. To back-fit IVHM onto a legacy component, instead of retro-fitting, it is usually more feasible to upgrade with a new component which the IVHM requirement is defined with a new system acquisition. [12]

To facilitate the IVHM system implementation process, a needed architecture would be those can support fully developed IVHM technologies and processes that can be implemented now and as well as yet-to-be developed capabilities. A new conforming IVHM component (on- or at-system) would be able to plug-and-play into an existing IVHM network. The architecture would also allow system reconfiguration to be carried out with ease, and in the process a new IVHM data processing chain could then be formed by linking or bolting-on multiple connected components. Energy harvesting IVHM components will in particular post an additional challenge to a supported architecture. This type of device has an intermittent operable time characteristic. In order to employ energy harvesting based devices in an IVHM application, an architecture that also allows fast re-connection of IVHM components to the network and rapid setup of a pre-defined IVHM data process would be desirable.

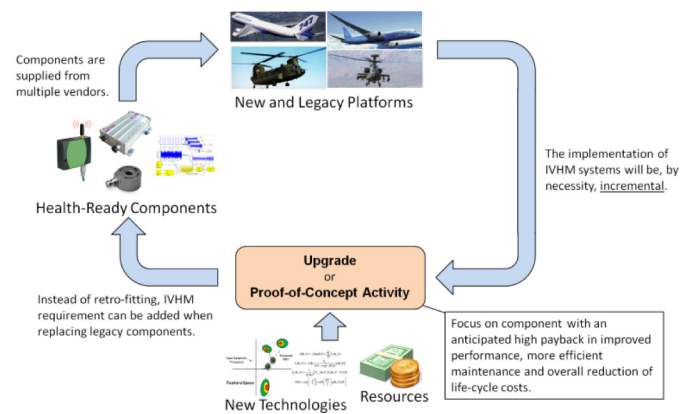


Figure 1. IVHM System Implementation Process.

Open Systems and Data Strategy

The open systems concept is an essential element of IVHM. It refers to the design of hardware and software based on industry and government standards and widely accepted data models. Its goal is to allow interoperability between multiple vendors' IVHM components through the use of standardized interface and data model [13].

In the big picture, an IVHM system is effectively parts of an end-to-end maintenance operation (i.e. asset health & usage, reliability and maintenance operations), which IVHM belongs to the asset health & usage sub-domain. IVHM data is essentially required to be shared across maintenance organizational boundaries [14]. Hence, in addition to data compatibility among IVHM components, the IVHM data model is also required to be consistent with the rest of the organization's data (i.e. reliability and maintenance domains).

Open System Architecture for Condition Based Maintenance (OSA-CBM) is a needed open standard which in particular addresses interoperability and enterprise systems integration of the IVHM applications [15, 16, 17]. OSA-CBM is divided into the interface definition and data model. These specifications are defined using the Unified Modeling Language (UML) and are intended to be technology neutral. They can be mapped into various programming languages and middleware technologies.

The OSA-CBM data model is based on the concept of metadata. The aim is to have data that are always identifiable and traceable. Moreover, metadata enables IVHM data to be shared across maintenance-related applications. OSA-CBM DataEvent is the dynamic data related to IVHM events generated by an IVHM application such as measurement, manipulated or processed data, etc (see figure 2). Its metadata are used to identify and inform status of the data event. The DataEvent inheritance hierarchy is associated 6 abstract IVHM data processing functionalities. The functional definitions can be viewed as abstractions of data into knowledge with higher and higher abstraction level of information. The DA data type is an acquired sensor data which is formatted into a consistent form. The DA data is then transformed into one or more meaningful features, which are in the DM-type data format. The features are compared against expected values, and the resulting enumerated condition indicators are stored in the SDDataEvent class. The data in the HA, PA and AG formats are the data related to current health of the machine, predicted future failures and recommended action steps, respectively.

The interface specification describes how data events will be moved. It can also be seen as service offered by the IVHM applications in the Service-Oriented Architecture (SOA) sense. There are 4 primary types of interface to accommodate different purposes and technological capabilities: Synchronous, Asynchronous, Data Service and DataEvent Server. The Synchronous interface returns data with the call. It models stateless service functions as founded in SOA. The Asynchronous interface allows any number of IVHM applications to establish and maintain a two-way connection for the duration they need. The data is returned either on request, on alert or by push all. The latter pushes data to the connected IVHM components every time it collects data without the need for a request beforehand. The Data Service interface is for a one-way data input device. Two possible uses would be data storage utility and maintenance advisory receiver service. The DataEvent Server interface is similar to the return on alert or the push all modes of communication in the Asynchronous interface (see figure 3). Runtime data (un-) subscriptions are enabled in the Asynchronous and DataEvent Server interfaces.

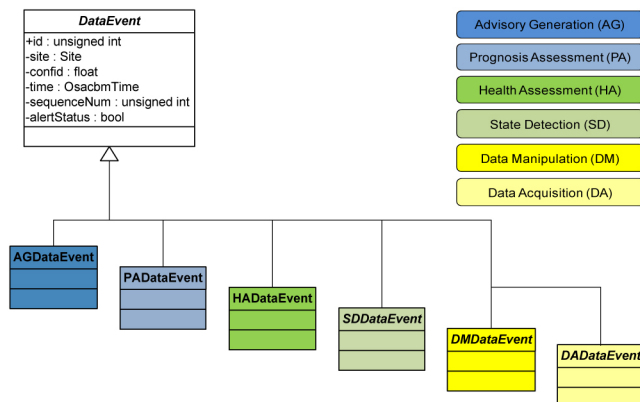


Figure 2. OSA-CBM's 6 Abstract Dynamic Data Event Models

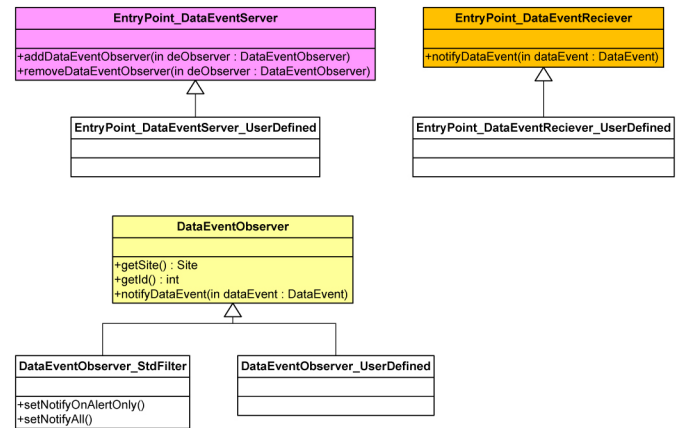


Figure 3. OSA-CBM DataEvent Server Interface

DESIGN GOALS

SOA dominates today's enterprise architecture. Despite its use for integrating enterprise applications, is it the best fit for PnP IVHM architecture? SOA focuses on implementing synchronous request/response interaction patterns to connect remote processes in distributed systems [20]. In the context of IVHM data process, it is the interface (or service) for requesting or polling for the data from network embedded components. The embedded IVHM components are connected through the interface to create a distribute IVHM system. It assumes the relationships between the embedded IVHM components are clear, and each know exactly where and when to request data. Consequently, every component is essentially coupled to each other. It can be seen that these technical characteristics contrast with the required data flow and construction of data process features described in 'IVHM Data Process' section. Moreover, the SOA approach will only work in a centralized environment. There is a need for a system administrator in the loop as significant efforts will be required to connect or re-configure the network embedded IVHM components.

If not SOA, then what will integration architecture be? An integration approach potentially meeting what required would be a data-centric architecture (see Figure 4) [21, 22, 23]. In this approach, system-level data are shared in and accessible from a global data space. Publish/Subscribe design pattern is keys in implementing the required IVHM data flow characteristics. Data servers publish data to the global data space, and on the contrary data receivers subscribe to interested data also from the same global data space. In this design pattern, data (un-) subscriptions are managed by the data space, and data are automatically delivered (or pushed) to the subscribed data receivers whenever the interested data (i.e. T1,...,Tn in figure 4) are updated. This will create a running network embedded IVHM system which is driven by stream of data. Note that it is possible for an embedded IVHM component to be both a data server and a data receiver. From now, the terms 'data server' and 'data receiver' can be referred to as 'embedded IVHM component'.

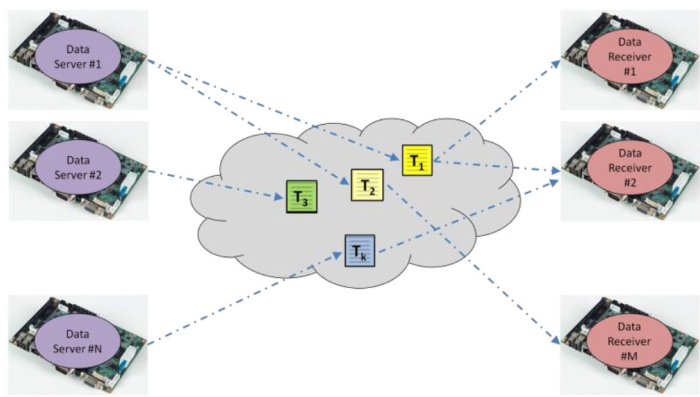


Figure 4. Publish/Subscribe Global Data Space

In terms of data process construction, data servers and data receivers are decoupled by the global data space. There is no direct connection between embedded IVHM components. A data receiver only needs to know what data it is receiving. It is not required to know where the data source is. Hence, only a minimal effort is expected if a new component is to be added to the system. Moreover, removing or replacing a component will not require rework for the other components as in terms of remote binding there is no coupling between them. It can be seen that the task of connecting or re-configuring a distributed IVHM system is significantly simplified by integrating the system around the shared data. The issues of how to implement the global data space concept will be discussed in ‘Publish/Subscribe Model’ and ‘Data Distribution Service Model’ sections.

A common (shared) data model is basis in data-centric architecture as the shared IVHM data must be understandable by all the network embedded IVHM components. Data events are identified (or distinguished) by the metadata contents which could be aggregately called *topic*. $T_1, \dots, T_n$ in figure 4 are examples of the topic subscription. The OSA-CBM data model is well designed and significantly relevant to IVHM data process (see ‘Open Systems and Data Strategy’ section). The OSA-CBM data model features both metadata and core IVHM data [16] (see figure 5). Its metadata fields describe information needed for data identification and sharing across IVHM and maintenance related applications. Examples of metadata are id - what data it is, site - which platform the data is for and time - when the data is generated. Hence, these make OSA-CBM DataEvent self-explanatory.

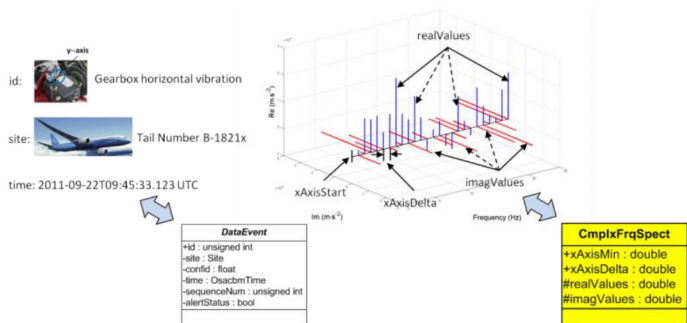


Figure 5. Example of OSA-CBM Metadata and Data: DMDDataEvent - CmplxFrqSpect

CmplxFrqSpect is an example how frequency spectrum data (e.g. FFT) is modeled in OSA-CBM DataEvent. realValues - real frequency components, imagValues - imaginary frequency components and xAxisDelta - a frequency resolution CmplxFrqSpect is derived from DMDDataEvent which is derived from DataEvent (see also figure 2). Consequently, CmplxFrqSpect itself is a type of DataEvent and hence inherits all the metadata fields defined in DataEvent. It is worth noting that any types of IVHM data can be related to one of derived types of OSA-CBM DataEvent. Together with well-defined metadata model, this makes OSA-CBM a complete data model and therefore a good candidate to be used in IVHM data-centric architecture.

DISTRIBUTED DATA MECHANISM - DESIGN CHOICES

Client/Server Model

Design: The first design choice for an architecture supporting IVHM data flow and construction of data process is to utilize OSA-CBM interfaces and implement embedded IVHM components as client/server applications [24] as illustrated in figure 6. Both OSA-CBM Asynchronous and DataEvent Server interfaces support the required data flow characteristics described in ‘Global Data Space’ section (i.e. push all or trigger by state changes) [16]. For simplicity, the DataEvent Server interface (with the least API methods) is implemented in this paper. The remote interface contains methods to connect distributed IVHM components, configure data filtering properties and notify/push updated IVHM data events (see also figure 3).

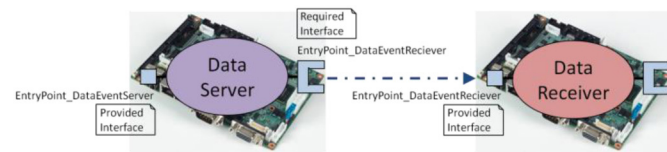


Figure 6. Client/Server Model

Analysis: The primary advantage of this approach is its simplicity. The DataEvent Server interface implements the Observer design pattern. The remote interface encapsulates the required publish/subscribe service functionalities (see figures 3 and 9) and can straightforwardly be implemented using Internet Communication Engine (ICE) [25] or other CORBA-like middleware [24]. In this paper, ICE is preferred to CORBA primarily due to its simplicity in implementation and support for object-oriented data models such as OSA-CBM. In this design, a global data space is not part of the architecture as a data server has straight access to the remote EntryPoint_DataEventReceiver interface of its associated data receivers. The notifyDataEvent(...) method is invoked to push an updated IVHM data to the receiving embedded components (see figure 7).

However, there are a number of shortcomings of implementing embedded IVHM components as client/server applications. First, a distributed IVHM data process is formed by creating multiple connections between data servers and data receivers thereby defeating the design goal of having a loosely coupled architecture. Second, a significant rework will be required if there is a change in

the distributed data process. The remote interfaces have to be bound or rebound for every affected IVHM component. This can be carried out by either dynamic bindings (see figure 7) or configuration file updates on the embedded components. Unless the changes are persistent, the former will have to be executed every time when the system starts up. The need of rework for every affected component creates the spaghetti effect making this design choice difficult to scale up. Finally, a highly automated configuration will not be possible as such configuration tasks assume a system administrator to be somewhere in the loop. The IP address and name binding of a remote interface are string based, and their meanings require a human interpretation which is needed in the setup of dynamic bindings and creation of configuration files.

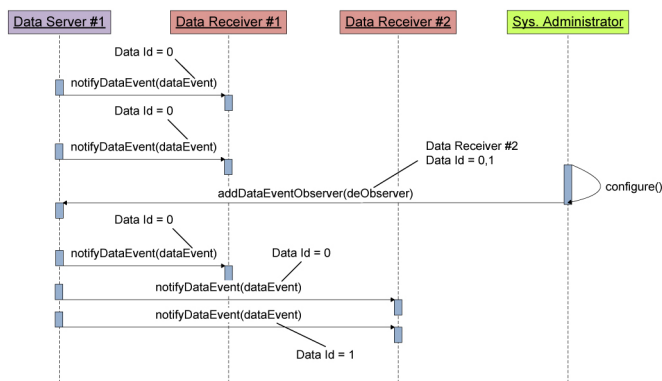


Figure 7. Client/Server Model - Data Mechanism

Publish/Subscribe Model

Design: Figure 8 depicts a second design for PnP enabled IVHM architecture. This design implements IVHM embedded components as client/server applications with OSA-CBM DataEvent Server interface as in the first design choice. However, the data server and receiver components are not directly connected. In this design, an additional publish/subscribe (pub/sub) server is required as part of the IVHM architecture [22,24]. The pub/sub server acts as a global data space and is responsible for managing data (un-) subscriptions and delivering updated IVHM data events to the subscribed IVHM components. Since data subscriptions are managed by the pub/sub server, hence only EntryPoint_DataEventReceiver needs implementation for the embedded IVHM components. The pub/sub server however will also require an implementation of EntryPoint_DataEventServer or other similar interfaces to provide remote methods for connecting distributed IVHM components to the data space and configuration of data filtering properties (see also figure 3).

Analysis: This design choice has advantages in terms of configuration scalability. The design implements the Publish/Subscribe design pattern. In contrast to the first design, an updated data is pushed to its subscribed data receivers through the pub/sub server (see figure 8) instead of directly from a data server. The data server and its related data receivers are decoupled by the pub/sub server. There are no direct connections between them. Therefore, adding or removing a component will not require rework in terms of remote bindings for the other related IVHM components. Only the

pub/sub server will require re-configuration either by dynamic binding (see figure 9) or configuration file update. This design approach makes IVHM system implementation easy to scale up.

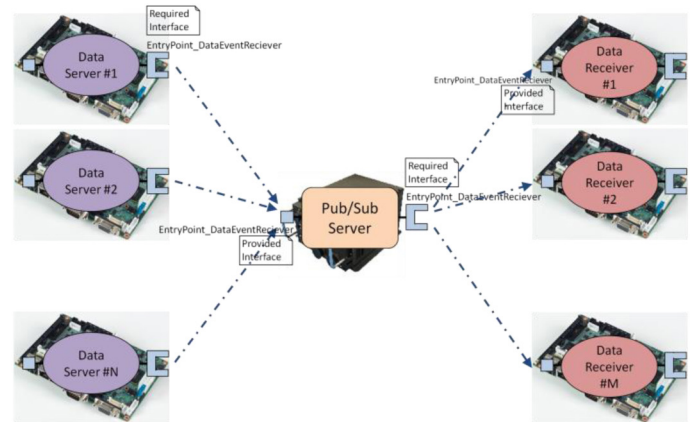


Figure 8. Publish/Subscribe Model

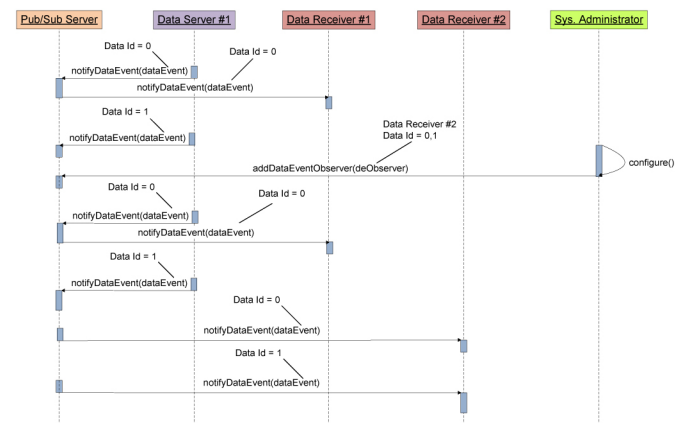


Figure 9. Publish/Subscribe Model - Data Mechanism

The remote interfaces EntryPoint_DataEventReceiver and EntryPoint_DataEventServer required for publish/subscribe functionalities can be straightforwardly implemented using ICE or other CORBA-like middleware. Alternatively, a specialized publish/subscribe middleware (e.g. IceStorm) can also be used to simplify the implementation [25]. It provides a commercial off-the-shelf (COTS) pub/sub server and a framework for defining data model. IceStorm is one that allows OSA-CBM data model, however it does not support content-based filtering as described in 'Design Goals' section. Hence, a data receiver is responsible to implement data filtering in the case of IceStorm. Alternatively, a unique topic name can be defined and used to associate with each data event id. There are other pub/sub middleware that support content-based filtering, but all non-research-prototypes do not allow a custom data model like OSA-CBM.

One disadvantage of this design is data latency. Data is indirectly delivered to a data receiver through the pub/sub server. This means the delivery time will be at least twice that experienced on the direct connection. In this approach, more reliance is on the pub/sub server, and what if the pub/sub server fails? Despite its advantages, the pub/sub server clearly represents a potential single point of failure (SPOF) in itself. Moreover, replacing the pub/sub server with another one for all data servers unless a new server is exactly the same