**VISTA Adaptive Maintenance v1.0.0 SDD Addendum**

The System Design Document (SDD) chief purpose is to depict an architecture decomposition of an implementation, such as an application or service. This architecture break down will enable managers, architects, developers, and SDD reviewers to understand the employed technologies, major design components and their relationships, message flow, and physical deployment. Other than providing the required information about the implementation, the main sections of an SDD should focus on three perspectives: Conceptual, Logical, and Physical.

# General Implementation Information

Name: VISTA Adaptive Maintenance

Type: Cloud-based Security, Auditing and Emulation

Version: 1.0.0

Product life cycle Critical Decision in VIP (CD1, CD2): CD2

## Brief Description (business and technical)

VistA’s Remote Procedure Call (RPC) Interface was built incrementally over two decades to support a variety of clinical and business clients. It is fundamentally insecure and lacks a comprehensive auditing mechanism.

VistA Adaptive Maintenance (VAM) is a Veteran-focused Integration Program (VIP) that moves management of this essential Interface from VistA to the cloud. This transfer enables a variety of security upgrades and functional migrations without changing either VistA or its clients in any way.

Specifically, VAM provides comprehensive, cloud-based monitoring and security of the RPC interface and migration of a representative subset of that interface to backward compatible, secure, cloud hosted microservices with a national footprint.

VAM will be hosted in production within the VA’s Enterprise Cloud (VAEC) using Amazon Web Services (AWS) and leverage the AWS CloudWatch service.

# Design: Perspectives, Artifacts, and Technical Specifications

In VAM, an “RPC Router” sends all RPC traffic into CloudWatch, the AWS COTS Application logging service where it can be processed and analyzed. In addition, the Router may re-direct select RPCs to cloud-based microservices allowing VistA functionality to be retired without effecting VistA clients (“emulation”).

## Conceptual Perspective Section

In the RPC Router, basic client and RPC identification is decoupled from how individual RPCs are secured, logged or emulated. A core RPC Engine performs basic RPC handling and delegates policy-based handling to processors (“RPC Handlers”) arranged in a chain of responsibility. In keeping with the chain of responsibility pattern, individual RPC Handlers may allow or prevent further processing of an RPC.

How emulation happens, the criteria used for logging and the policies applied for security are delegated to these RPC Handlers. In addition, more complex handling including raising alarms may be handled in COTS middleware delegated to by these Handlers. This approach allows for a variety of logging, security and emulation approaches in the same architecture while maximizing the use of third party and COTS facilities.

### Sessions

The RPC protocol is connection-oriented and synchronous with clients opening a connection to VistA and only making new RPC calls after receiving a reply from a previous call.

* Connection establishment and ending define a “client session” in the RPC engine and all RPC traffic on that connection is identified with that session.
* Clients log into VistA in different ways – there are Connection Proxies, CAPRI tokens, BSE tokens, Access Verify, SAML tokens. Each method is recognized by the Router and allows it to associate a client’s identity with the session. It is important to note that the Router doesn’t implement authentication – it merely notes how VistA responds to different sign on options and changes the client session appropriately.
* Session identity and details are passed into RPC Handlers along with a parsed version of an RPC
* An RPC Handler may signal the Router engine to end a session

### RPC Model

A comprehensive model for RPCs is at the center of VAM (“RPC Model”). VistA’s basic RPC description file, 8994, does little more than name RPCs and MUMPS functions for processing them. The Router and its Handlers require much more detail.

* Is the RPC for authentication (session), patient record access or basic meta data access?
* Does it read or change data?
* What types of arguments are permitted in an RPC request?
* How complex is the RPC?
* What domain does it belong to? Allergy, Pharmacy …
* What parameters and FileMan files does it involve?
* Is there an emulation service available for the RPC?

Most of this nuance is only used in RPC Handlers. The engine’s main task is to associate an RPC with its model and pass that model along with session information and the RPC itself to the Handlers. The use of a model allows the handlers to be data-driven and avoids the creation of a large volumes of procedural code.

There are over 3,500 RPCs today. As VAM expands, more and more are being modeled. Handlers are written to expect either no model or different degrees of modelling. For example, an RPC may be modeled sufficiently to allow for precise logging but not modeled sufficiently to enable a Security Handler designed to prevent illicit argument values. In VAM, it’s important not just to identify whether an RPC is modeled but also what level of modeling it has received.

3,500 is a daunting number. To aid in model expansion, VAM has a series of model generation tools that analyze MUMPS and client code as well as logs of RPC traffic to derive definitions. Ultimately, all RPCs must be modeled to allow the full range of handlers to operate on them.

The use of the model for logging is detailed in the Data section below.

## Logical Perspective Section

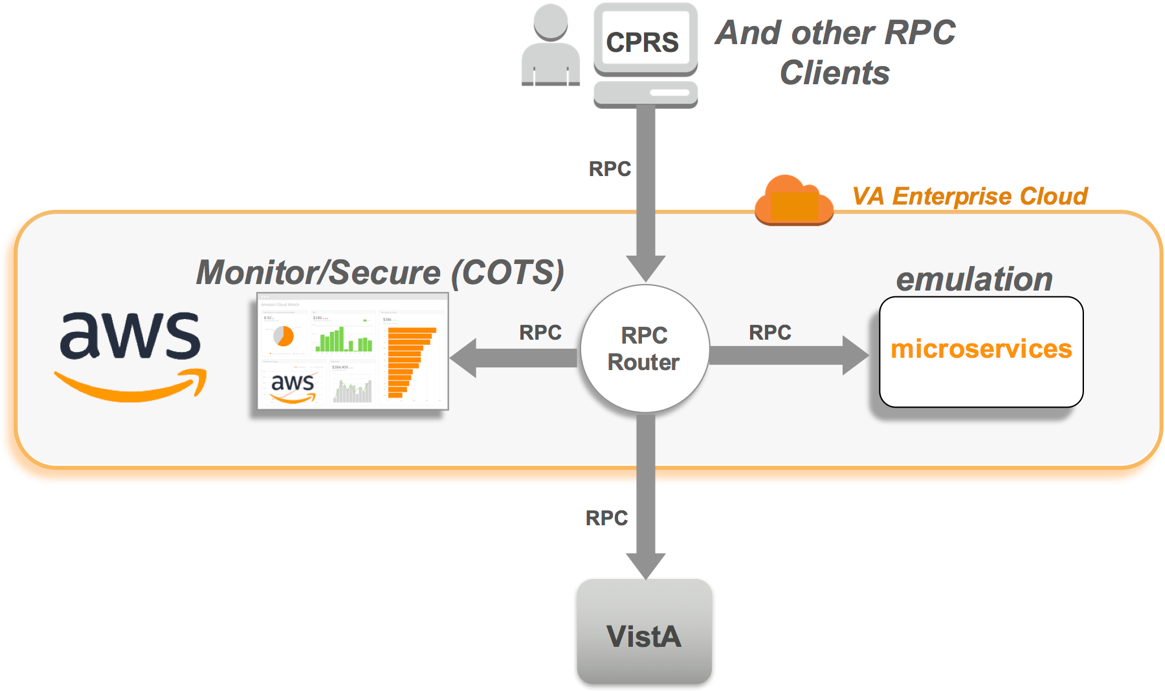
The decoupling of the core Router engine and specific RPC Handlers is detailed in the Conceptual Perspective Section.

## Physical Perspective Section

The RPC Router is deployed in two ways, Monitor mode and Intercept mode.

In Monitor mode, the Router processes a copy of RPC traffic from a network TAP (“Terminal Access Point”), leaving clients to connect directly to VistA as they do now. This mode supports full monitoring including alarming of all activity and provides an historical record.

In Intercept mode, the Router sits between the VistA Clients and VistA. The clients are redirected to talk through the Router instead of going to VistA directly and this redirection can be handled in network routing tables and doesn’t involve changes to the Clients or VistA. In addition to monitoring, sitting in between Clients and VistA enables complex RPC handling including preventing illicit RPCs getting to VistA and re-directing certain RPCs away from VistA and into microservices. Intercept mode is pictured below.



As the RPC Interface is a layer 7, proprietary, ASCII based synchronous protocol sent over TCP connections, both Monitor and Intercept mode involve connection-oriented traffic management and load balancing, facilities available in AWS-based VAEC. As the picture above indicates, the Router and its RPC Handlers run on Virtual Machines in the VAEC, while VistA and its clients remain unchanged.

The RPC Router and its handlers are implemented in Javascript running in node.js inside orchestrated containers. The logging handlers use AWS Cloudwatch or Syslog and are detailed in the data section below. These handlers not written to use CloudWatch or Syslog directly. By using a portable logging API, the same handlers can run on both.

Emulation handlers store data in a Cloud-based Mongo database. In order to test and demonstrate Emulation microservices, VAM also provides a basic React.js based Web Client that can use the services independently of RPC based clients such as CPRS.

# Data Architecture: Persistence Storage and Middleware Environment Specifications

Logging of RPC Traffic is a core VAM feature. As both the mechanism and approach to logging is decoupled in RPC Handlers, different approaches are supported, each named by the name of its Handler.

As its name suggests, the “*Modeled RPC Logger*” uses definitions of RPCs in the RPC model to partition traffic into four categories. Each category leads to a different log. The four types are detailed in the table below.

|  |  |  |
| --- | --- | --- |
| **Log/Category** | **RPCs Covered** | **Comment** |
| Session | Login, User Identity | Each of VistA’s mechanisms for logging in and user identity is modeled. The login mechanism establishes a context for subsequent RPC calls by a user. A session ends when a User disconnects |
| Patient | Patient Record (PII/PHI) | Patient RPCs Select, Read and Change a Patient’s Record in VistA. Covers clinical and non clinical data. It is key for effective auditing of traffic to separate Patient Record access from other activity. |
| Meta | Non PII/PHI with specific purpose | Examples include getting the time, looking up a code for a problem or a drug. These are both the majority of RPCs in kind and in use. |
| Raw | General purpose allowing access to both PII/PHI and other data | Some RPCs provide raw access to any data in VistA’s database. |

Unmodeled RPCs are ignored by this Handler. They are logged into a fifth log, “Other” by another handler, the *Unmodeled RPC Logger*.

While VAM currently focuses on the VAEC and AWS CloudWatch, its de-coupled handlers allow for other mechanisms. As handlers can act in tandem or be switched in and out, a Syslog based RPC Logger could run in parallel or instead of the CloudWatch based logger.

CloudWatch provides two facilities that ease VAM deployment. Like all cloud-based services, management of persistency – how long to keep various logs and in what form – is a core feature. Support for alarming and different ways to signal conditions is also of use in VAM. Handlers can raise alarms which can be propagated using generic cloud services. The use of generic Cloud services inside handlers is expected to expand as the project proceeds.