TACOM

Mobility and Firepower for America's Army

Vetronics Application Programmer Interfaces (API's)

Jeff Jaster

Electronic Architecture Team

Email: jasterj@tacom.army.mil
(810) 574-5106 / DSN 786-5106
Fax (810) 574-5933

U.S. Army Tank-Automotive RD&E Center (TARDEC)
Vetronics Technology Area
(AMSTA-TR-R, Mailstop 264)
Warren, MI 48397-5000

31 May 2001

UNCLASSIFIED

Report Documentation Page

Report Date 29May2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Vetronics Application Programmer Interfaces (APIs)	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Jaster, Jeff	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) U.S. Army Tank-Automotive RD&E Center (TARDEC) Vetronics Technology Area (AMSTA-TR-R, Mailstop 264) Warren, MI 48397-5000	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es) NDIA (National Defense Industrial Assocation) 211 Wilson BLvd., Ste. 400 Arlington, VA 22201-3061	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes Proceedings from 2001 Vehicle Technologies Symposium - Intelligent Systems for the Objective Force 29-31 May 2001 Sponsored by NDIA	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 20	

Agenda

- What is an API?
- API Based Software Reference Architecture
- Operating Environment (OE) API
- Adaptable Graphics Interface Language (AGIL) API
- Weapon Systems Mapping Services (WSMS) API
- Terrain Services API
- Performance, Analysis & Measurement API
- Station Management API

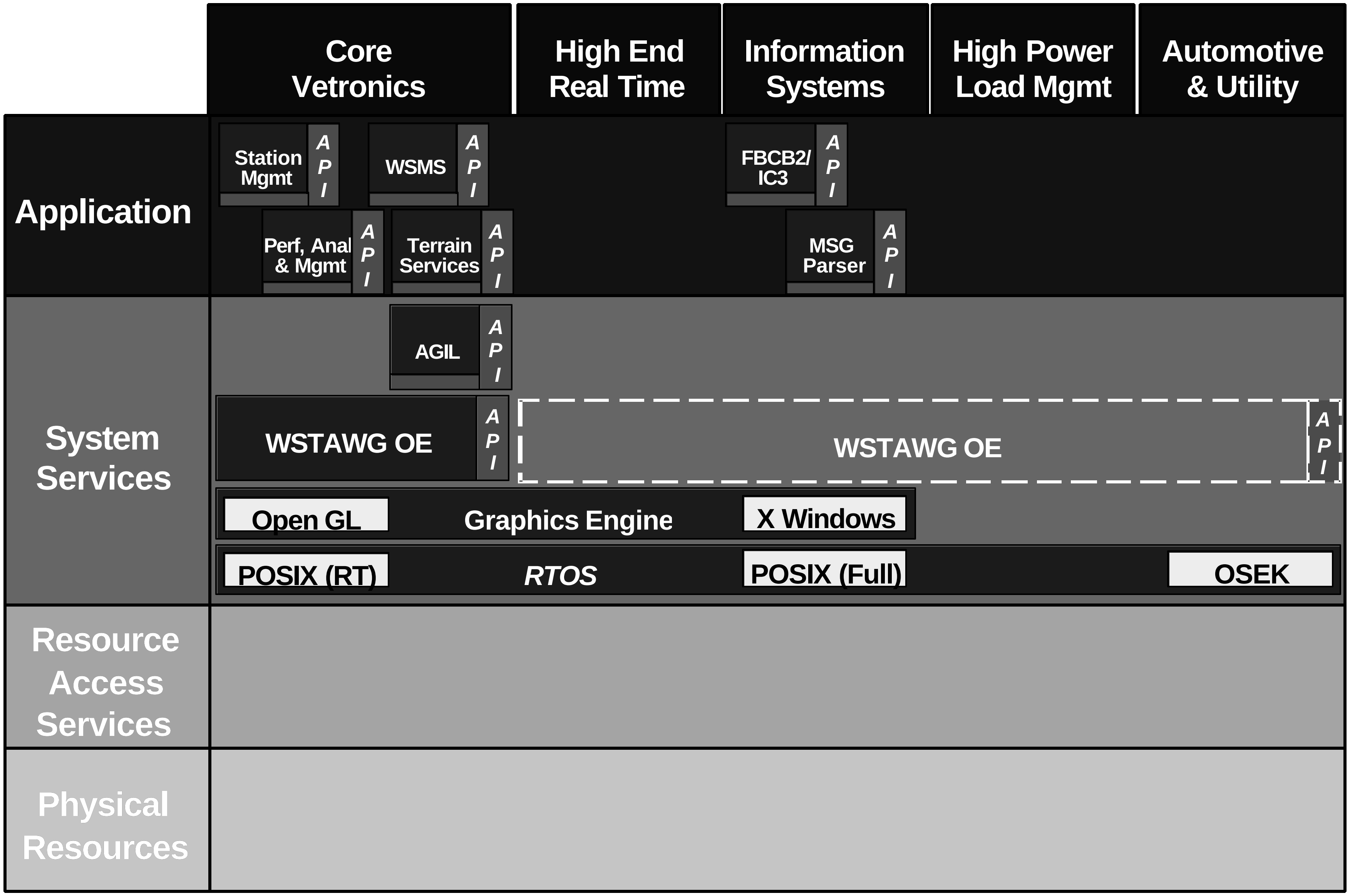
What is an API?

- Provides a service that receives and operates on some type of data, and returns data or some type of status regarding the success or failure of the service attempted.
  - Developed in a non-proprietary and open system format.
  - Provides flexibility where ever possible.
  - Layered and focused on interfaces.
    - Provides traceability from API to defined system requirements.
    - Designed for reuse and interoperability (define physical/logical interfaces).
      - Defined to isolate dependencies, to ease porting.
      - Defined to be adaptable in order to map to a variety of implementations.
    - Defined such that they can be replaced by emerging standards as they mature and are accepted by industry and DoD.
    - Designed for testability (carry through conformance/validation requirements).
  - Include industry, academia, and standards bodies to the degree possible when defining new API's.

API Based Software Reference Architecture

- Concentrates on interface definition by identifying applicable APIs and standards for physical and logical interfaces.
  - Utilizes SAE GOA model as a clear concise framework to partition capability.
  - Concentrates on interfaces to achieve interoperability, not products.
- Benefits of using an API based Architecture:
  - Promotes reuse at multiple layers.
  - Minimizes application impact from insertion of new technologies.
  - Facilitates interoperability through the identification of unambiguous interface definitions.
  - Enables plug and play capability not only at the resource access services layer (hw/drivers), but at the system services and application layers as well.

Example Software Architecture based on the Vetronics Reference Architecture



Operating Environment (OE) API

- OE is the mechanism to facilitate application porting among platforms.
- Supports the development of portable, reusable applications.
  - Support the development of embedded applications in heterogeneous distributed real time environments.
  - Provide semantic/behavioral correctness across varying OS and hardware platforms with predictable performance.
  - Provide extensibility and scalability to suit varying platform requirements.
  - Support distributed applications integration environments.
    - Communication and synchronization mechanisms developed to support the relocation of OE applications among processors and LRUs within a system without requiring the modification of application software.
  - Support the development and interoperability of applications in multiple programming languages (Ada 83, Ada 95, C) utilizing distinct OE vendor implementations.

Evolution of the OE Concept

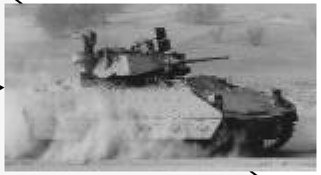
Product: **Standard Software Module (SSM)**
Contract: Standard Army Vetronics Architecture (SAVA)
Contractor: FMC, GDLS, TI, General Electric
Time: 1988-1992

Product: **Combat Vehicle Operating Env (CVOE)**
Contractor: Raytheon

Sensors
FLIRS

Product: **Bradley OE**
Contractor: UDLP-San Jose

M2A3



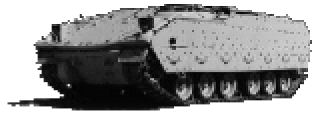
Product: **Vetronics RT Operating Services (VRTOS)**
Contractor: TACOM
Time: 1995-1996

VSIL

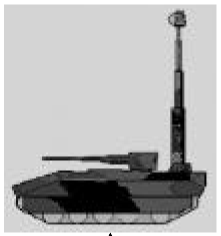
Product: **Crusader OSS / RTCOE**
Contractor: UDLP-Minneapolis

Crusader

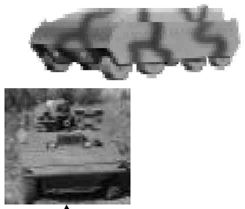
VTT STO



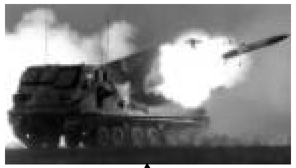
FSCS/Tracer



CAT/RF ATD

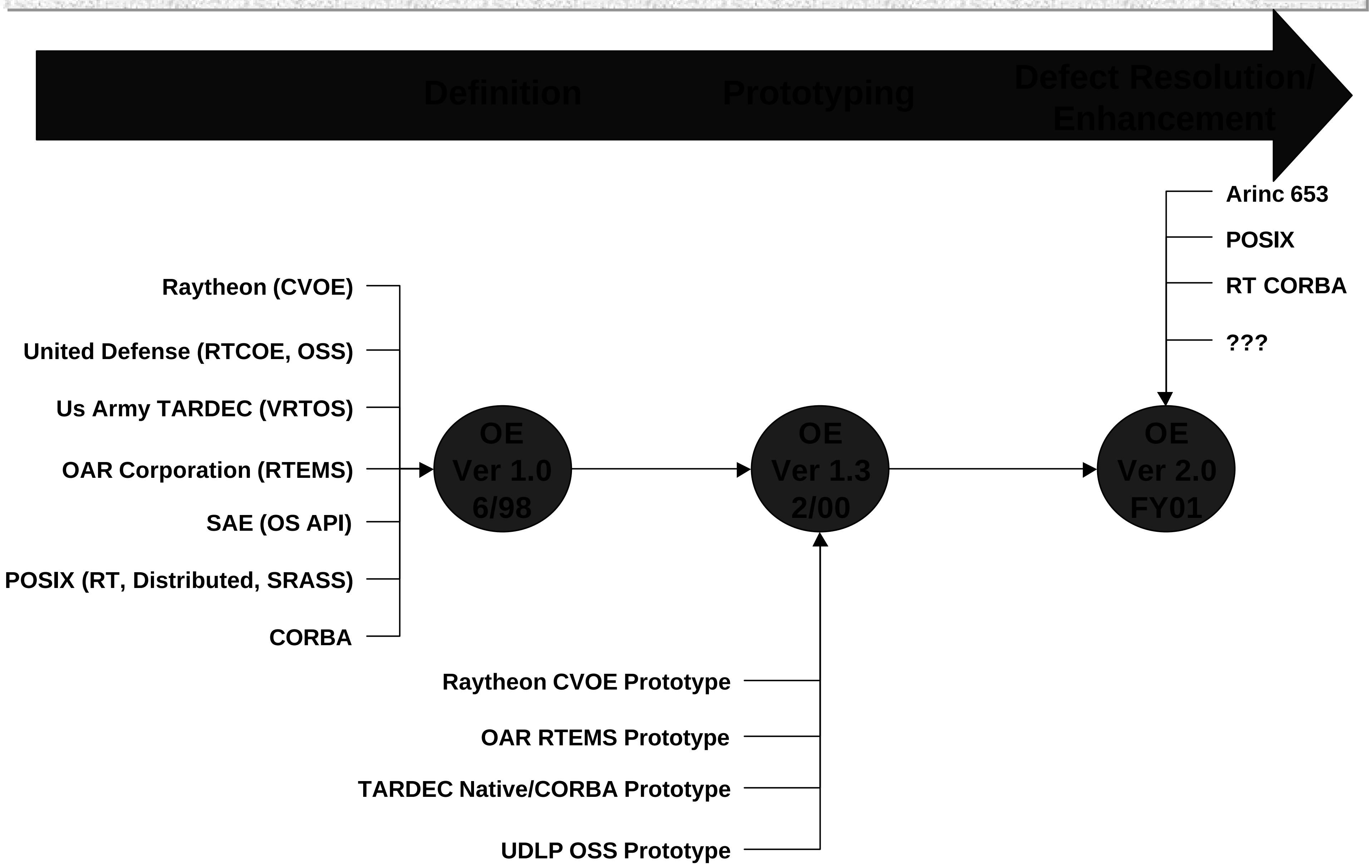


MLRS

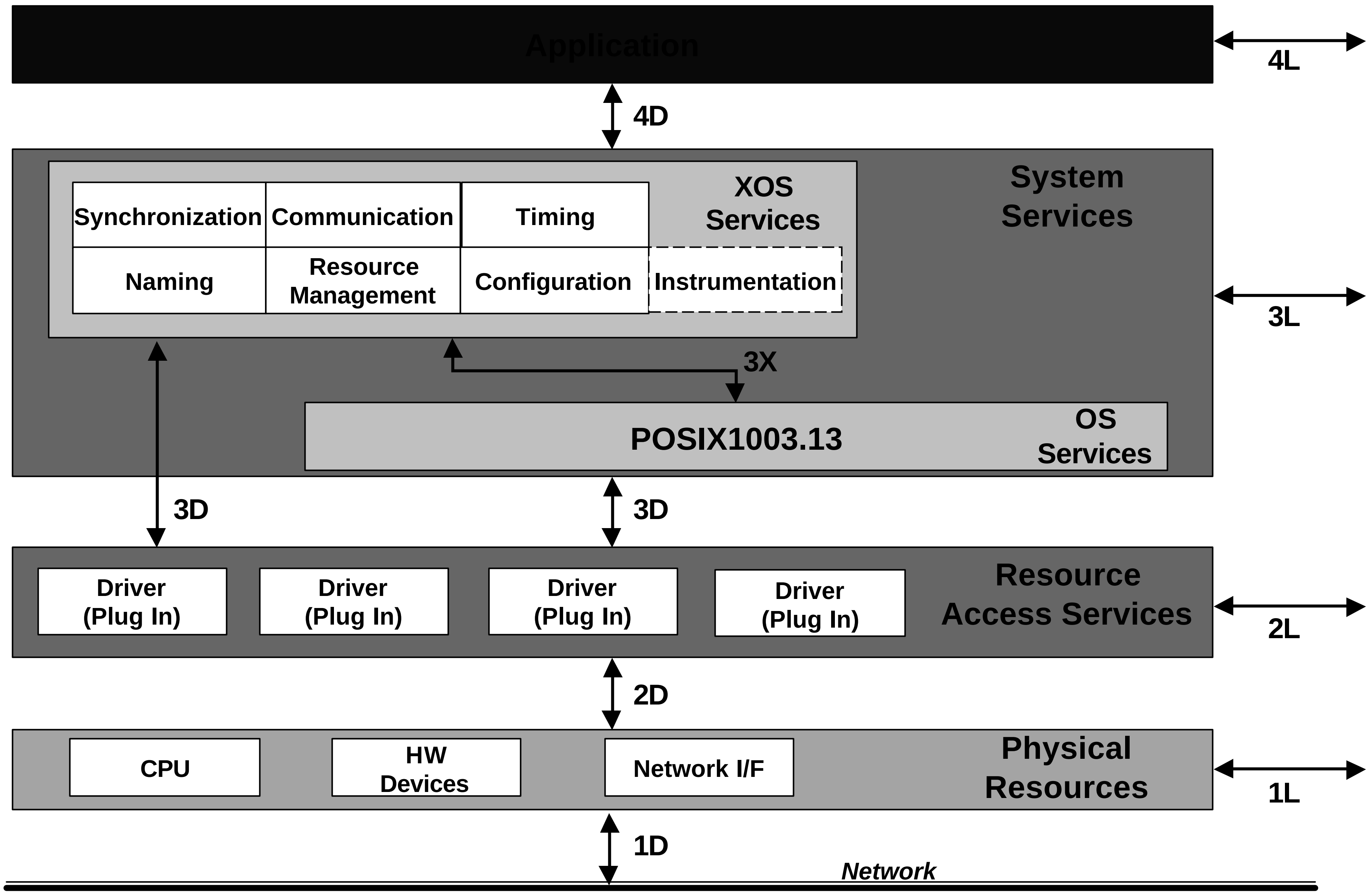


Product: **WSTAWG OE API**
Contractor: IPT
Time: 1997-Current

OE API Evolution



OE API Services



OE API Components

- API Specification
  - Language Independent Specification (LIS).
  - Based on CORBA Interface Definition Language (IDL).
- Language Bindings
  - Ada 95 and C Language bindings.
- Binary Encodings
  - Enables object access among distinct OE vendor implementations.
  - Based on CORBA Common Data Representation (CDR).
  - Provides opcode definitions and rules for interchange.
  - Provides rules for object registration and naming.
  - Provides protocol mappings (currently TCP/IP and Implementation Defined).
- Configuration
  - File structure and general syntax.
  - Partition structure and syntax.

Adaptable Graphics Interface Language (AGIL) API

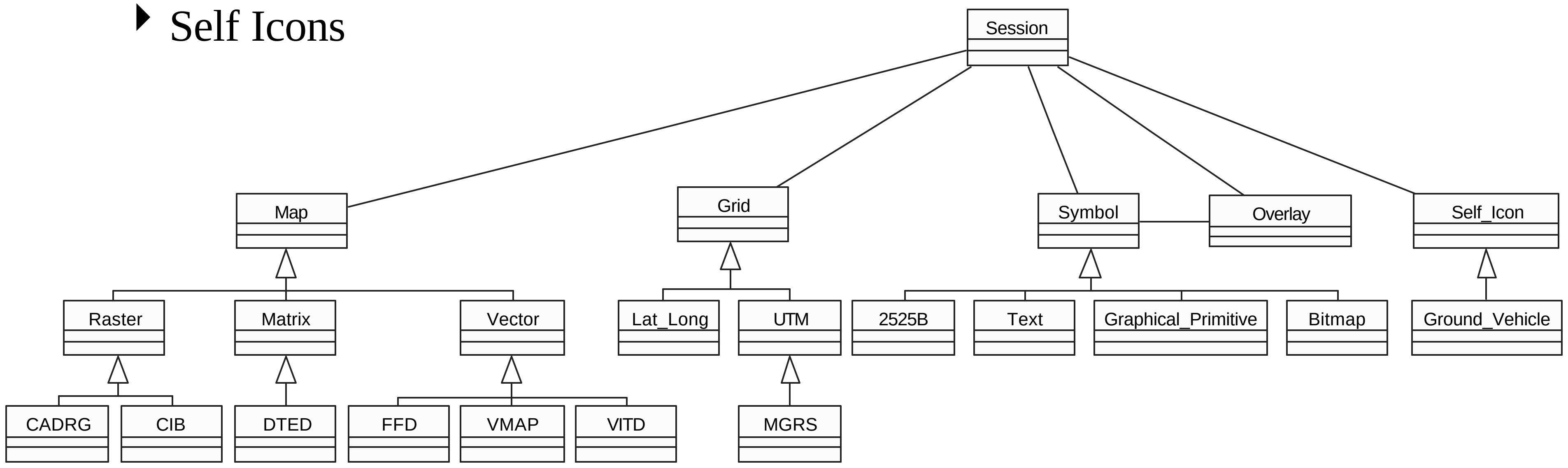
- Isolates the graphics engine hardware and software dependencies from the application software.
  - Defines an interface between the system software application and a graphical engine.
- The AGIL logical interface is defined via a language binding to a set of packages:
  - AGIL Primitives - Defines primitive data types/objects used in AGIL.
  - AGIL Device - Defines the core AGIL component, encapsulating application and graphics independence.
  - AGIL Event - Defines the primary mechanism utilized for interconnecting applications with the graphical device subsystem.
  - AGIL Drawable - Defines a set of complex graphical objects built utilizing the AGIL device API for development of an application SMI.

Weapon Systems Mapping Services API

- Defines a set of detailed concepts, functionality, and interfaces required to support the development of mapping applications, while addressing underlying architectural support issues of embedded real time systems (e.g. multitasking, reentrancy, and blocking/non-blocking execution).
- The WSMS Architecture has been developed to provide:
  - Application ***scalability***, such that groups of functionality and differing data standards and formats can be supported to meet varying system requirements.
  - Application ***extensibility***, such that future upgrades can be easily incorporated to provide for advances in mapping system technologies.
  - Support for the development of weapon system mapping implementations meeting ***real time embedded performance*** and integration requirements.
  - Support for the development of mapping implementations and applications within varying application architectures using ***varied graphics subsystems, hardware, software, and programming languages***.

WSMS API Architecture (Class Diagram)

- Developed as a set of abstract and concrete classes to implement specific functionality in the areas of:
  - Raster Maps
  - Vector Maps
  - DTED Maps
  - Grids
  - Overlays
  - Self Icons

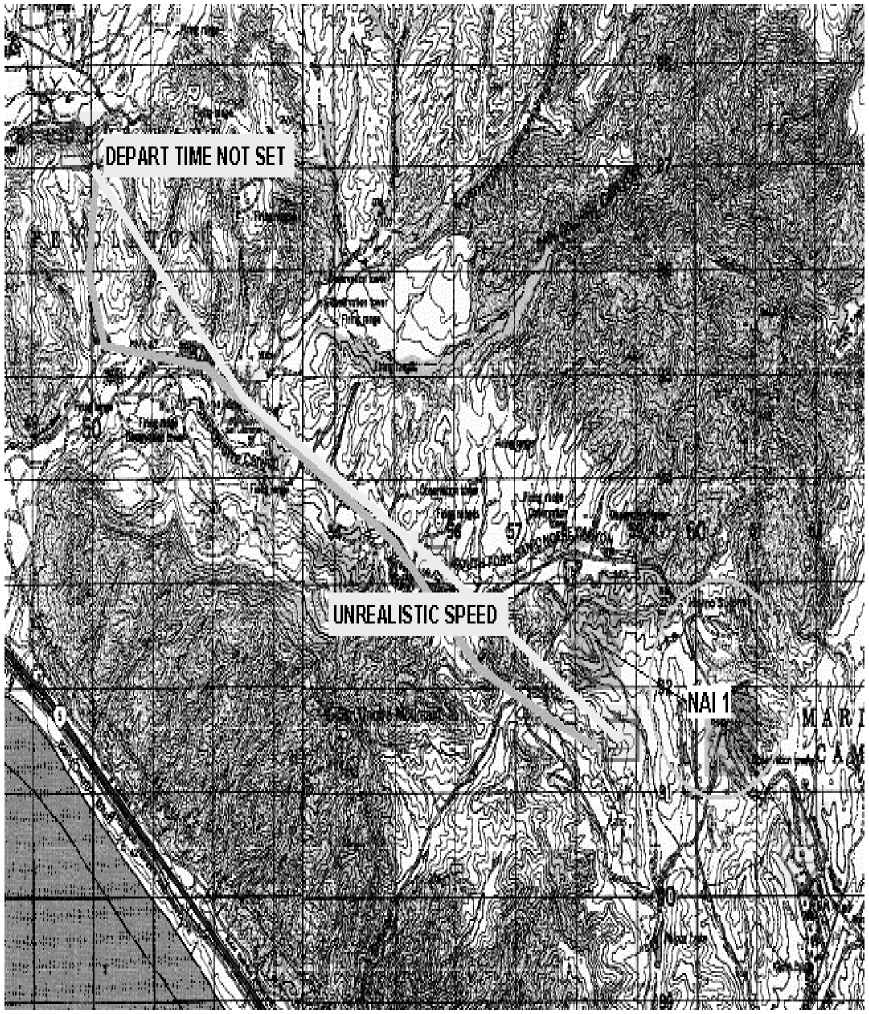


WSMS API Components

- API Specification
  - Language Independent Specification (LIS).
  - Described using a modified Backus Naur Form (BNF) notation.
  - Abstracts graphical and OS dependencies:
    - Graphics concepts are abstracted via a graphical implementation layer (GIL).
    - Underlying graphical engine is not assumed (e.g. could be X, OpenGL, Win32, DirectX, …).
- Language Bindings
  - Ada 83, Ada 95, and C++ Language bindings.

Terrain Services API

- Provides an interface to perform critical terrain analysis functions independent of operating environment and application developer.
- Developed to maximize:
  - Open-interfaces
  - Component reuse
  - Operational flexibility
- Primary Functions:
  - Path Analysis
  - Range Analysis
  - Terrain Categorization
  - Line of Sight



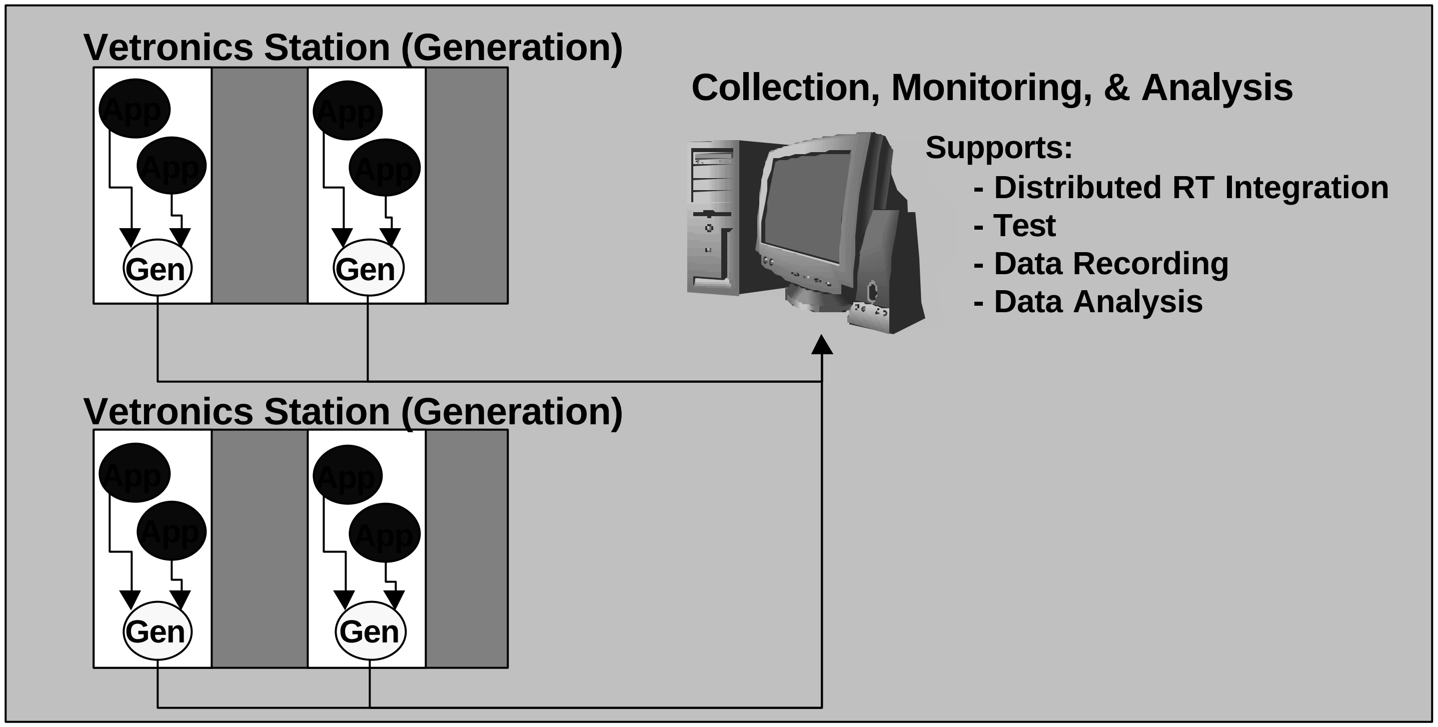
Developed in conjunction with ARDEC based upon a portion of the Future Fire Decision Support Software (F2DSS)

Terrain Services API (Cont.)

- Path Analysis
  - Calculation of mobility corridors.
    - Determination of choke points or restrictions identification along a path.
    - Calculation of travel times along a path.
- Range Analysis
  - Computation of minimum time, route and location to intercept a moving object.
  - Identification of possible locations at some future time for moving objects.
- Terrain Categorization
  - Categorization of terrain with client selectable parameters.
  - Identification of areas within given parameters.
- Line of Sight
  - Computation of visible line of sight from a location within an area.
  - Computation of direct fire weapons line of sight area from a location.
  - Identification of optimal observation points within a given area.

Performance, Analysis & Measurement API

- Extensible tagged event/sequence generation, recording, and monitoring system mapped to the "intelligent taxonomy" to measure defined mission scenarios/tasks (end to end).
- Primary Functions:
  - Internal – Generation Component.
  - External – Collection, Monitoring, & Analysis Component.


Vetronics Station (Generation)
App
App
Gen
Collection, Monitoring, & Analysis
Vetronics Station (Generation)

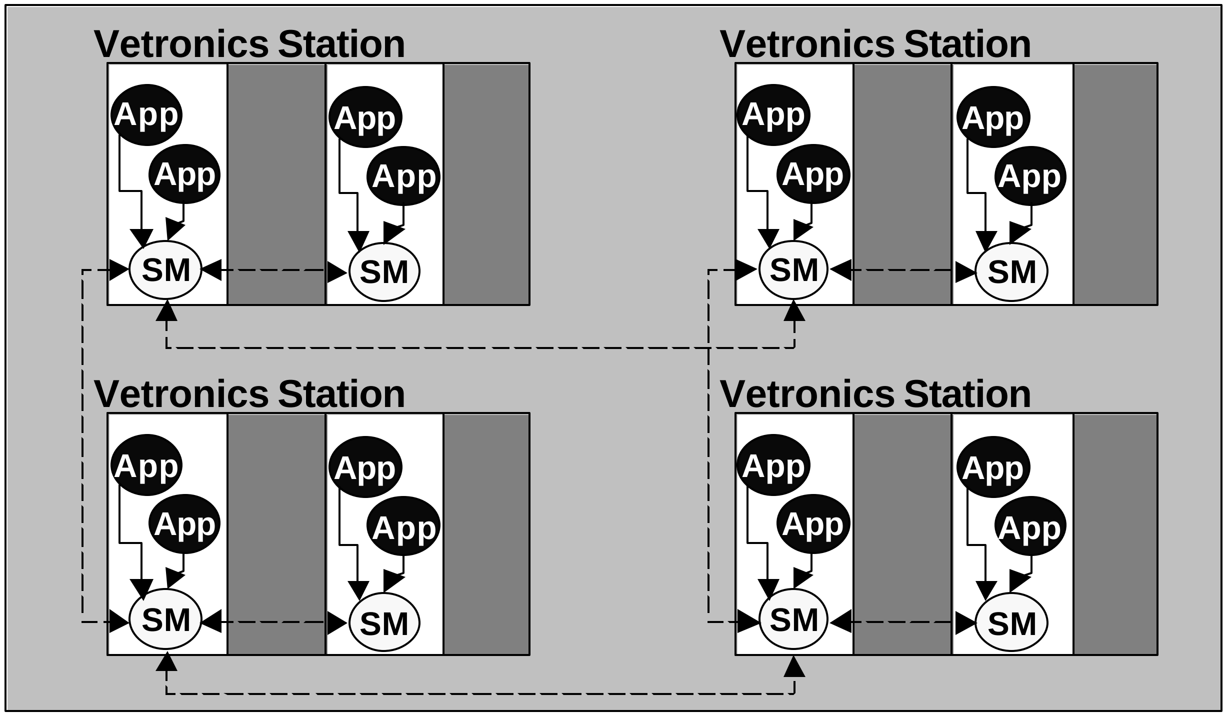
Supports:
- Distributed RT Integration
- Test
- Data Recording
- Data Analysis

Performance, Analysis & Measurement (Cont.)

- Generation Component (Internal)
  - Application interface is provided as an OO API with Ada 95 and C++ bindings.
  - Provides the ability to extend the base message to include additional architecture and application-defined data fields.
  - Provides the ability to control the routing and destination of output message data (internally for recording or externally for monitor/analysis).
- Collection, Monitoring, & Analysis Component (External)
  - Capture/Monitoring Functions:
    - Provides the ability to capture directed message data from the system in real time.
    - Provides the ability to retrieve/view/filter recorded data/messages from the target system for analysis.
  - Analysis Functions:
    - Provides the ability to view recorded sequences (threads) as an end to end set of actions.
    - Provides the ability to measure timing between discrete events.

Station Management API

- Provides a conceptual "software backplane" to facilitate application integration and system upgrade.
- Primary Functions:
  - Application loading and startup synchronization.
  - Application runtime synchronization.
  - Application subfunction health monitoring.



Station Management API (Cont.)

- Architecture
  - A station manager is resident on each computing node in the system, with one station manager being designated as the master for the station.
    - The station master provides for the synchronization of each node in the system.
    - The station manager provides a base concept/set of operations for a system state and mode and hardware registry which can be extended to a system specific context as required for the application.
- Application loading and startup synchronization:
  - Provides the ability to interpret a static configuration file in order to dynamically load/start subfunctions among the computing elements within the Vetronics stations.
- Application runtime synchronization:
  - Provides the ability for application subfunctions to synchronize at defined synch points.
- Application subfunction health monitoring.
  - Provides the ability to monitor the application subfunction health.
    - Provides the ability to report health anomalies to the system application.