Space Telecommunications Radio System (STRS) Architecture

Tutorial Part 1 - Overview

Glenn Research Center

November 2011

STRS Architecture

- STRS Background
- STRS Hardware & Software Structure
- STRS Infrastructure APIs
- STRS Application APIs
- STRS Configuration Files
- STRS Reference Documents

STRS Background

STRS Goals and Objectives

- Applicable to space and ground missions of varying complexity.
- Decrease the development time and cost of deployed capabilities.
- Increase the reliability of deployed radios.
- Accommodate advances in technology with minimal rework.
- Adaptable to evolving requirements.
- Enable interoperability with existing radio assets.
- Leverage existing or developing standards, resources, and experience.
- Maintain vendor independence.
- Enable waveform portability between compliant platforms.
- Enable cognitive radio concepts.

STRS Solution: Software-Defined Radio (SDR)

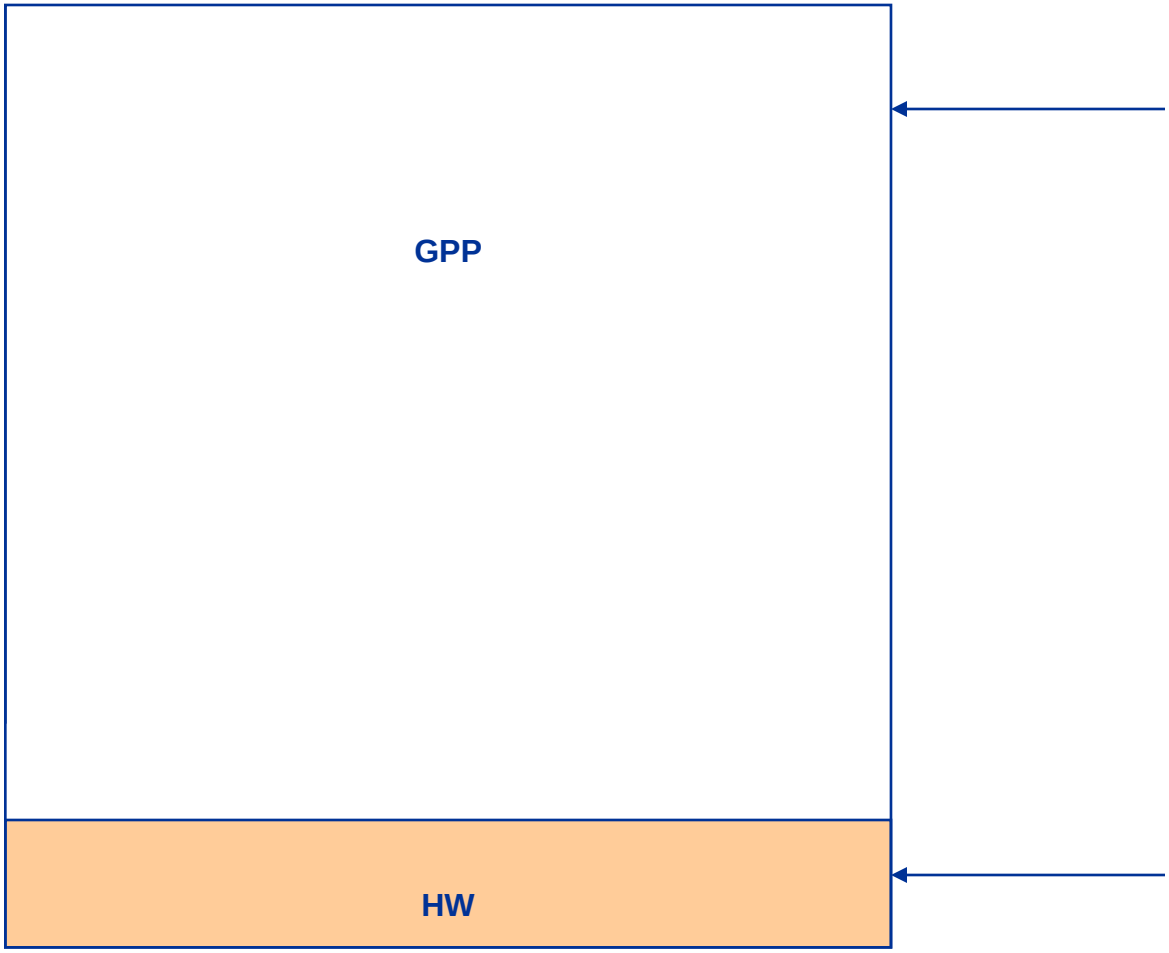
- SDRs are commonplace in commercial and military industries.
  - accommodates advances in technology
  - enables cognitive radio concepts
- SDRs allow encapsulation of functionality.
  - allows multiple vendors to work on different parts of the radio at once
  - allows updates to one part not to affect the other parts of the radio
  - allows portability
- Software design and implementation processes may be leveraged to lower risk and increase reliability

STRS Background

- SDRs present unique challenges in space.
  - Radiation environment
  - Temperature extremes
  - Autonomous operation
  - Size, weight, and power (SWaP) limitations
  - Timescale of deployments
  - Lengthy development cycles
- JTRS/SCA and OMG/SWRADIO were investigated
  - including CORBA was too cumbersome due to SWaP
  - including an XML parser was too cumbersome due to SWaP
  - SCA's XML configuration files were too complex for our needs
  - Used Platform Independent Model (PIM) as a starting point for STRS API design
- Decided to allow a C language interface to minimize SWaP

STRS Hardware and Software Structure

SDR Signal Processing Hardware



- General Purpose Processor (GPP) typically contains and executes the Managing Software enabling Software Defined Radio functionality

- Specialized Hardware contains and executes Application Software (i.e. firmware) enabling higher rate processing within the Software Defined Radio (e.g. FPGA)

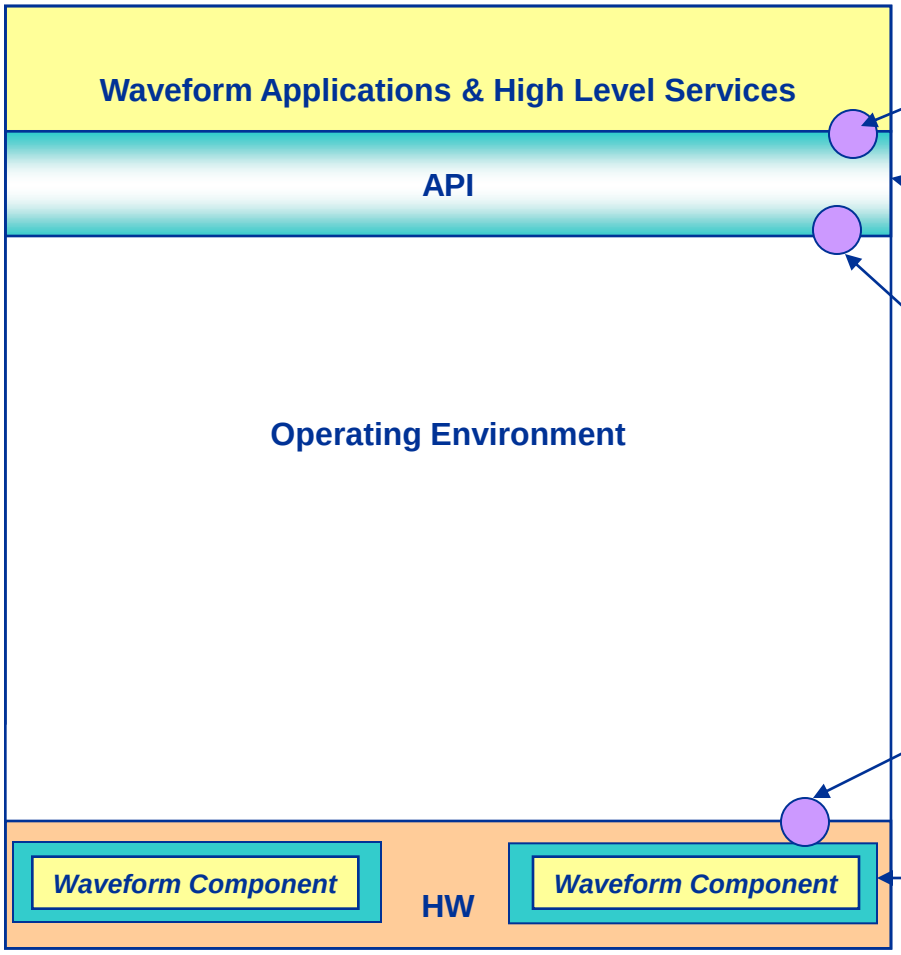
Waveform Application Decomposition

- Low rate waveform application components allocated to processor based signal processor. High level services used to control and monitor application software
- Operating Environment (OE) provides access to processor and specialized hardware for waveform processing
- High rate waveform application components allocated to specialized processing hardware

Waveform decomposition driven by designers, not architecture

STRS Open Architecture

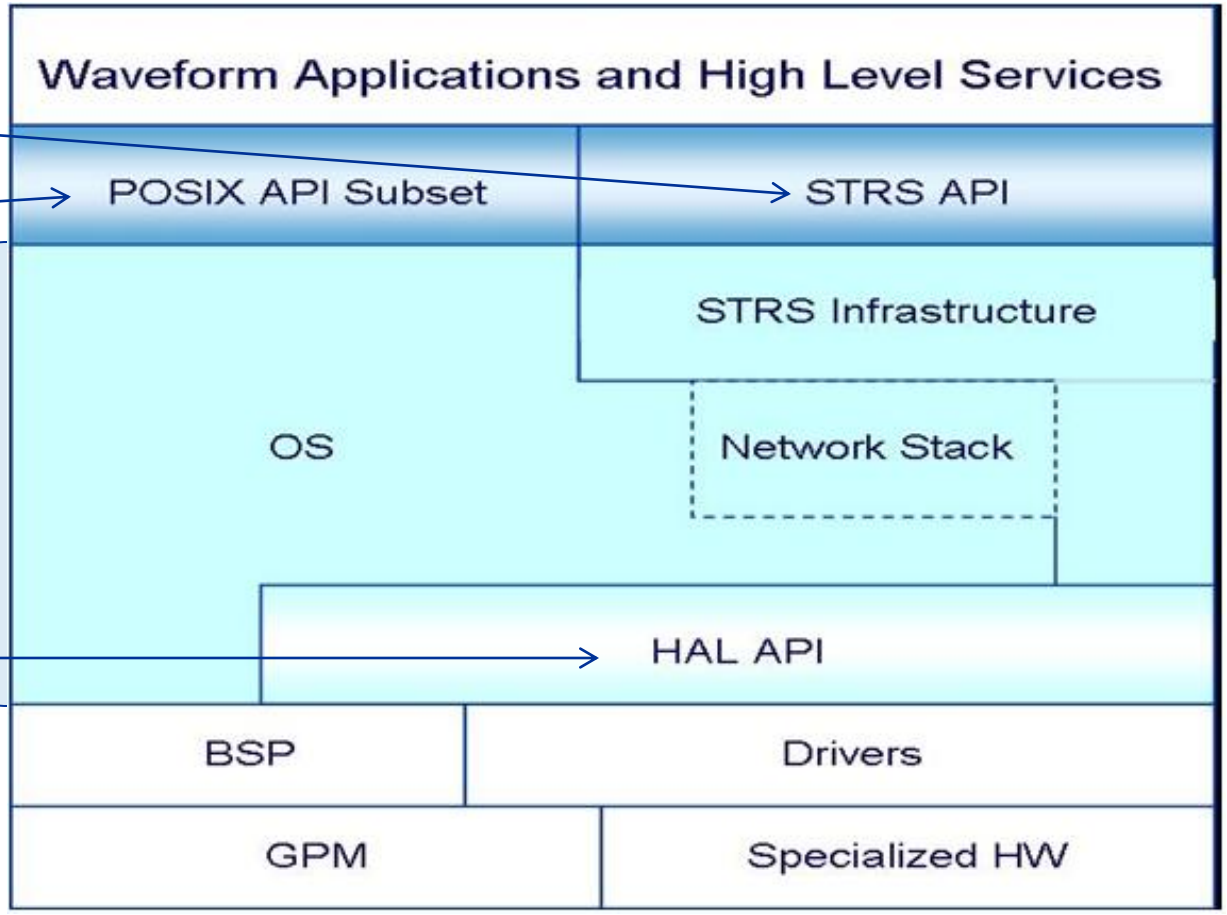
Waveform Application API and Hardware Abstraction



- Waveform designers access specified API to access processor and implements radio controls in the GPP
- APIs specified by architecture separate the waveform from the Operating Environment for waveform portability
- Operating Environment provides published interfaces (API) services and hardware abstraction control to waveform
- Hardware Abstraction Layer (HAL) also provides wrapper to help port HDL software among platforms

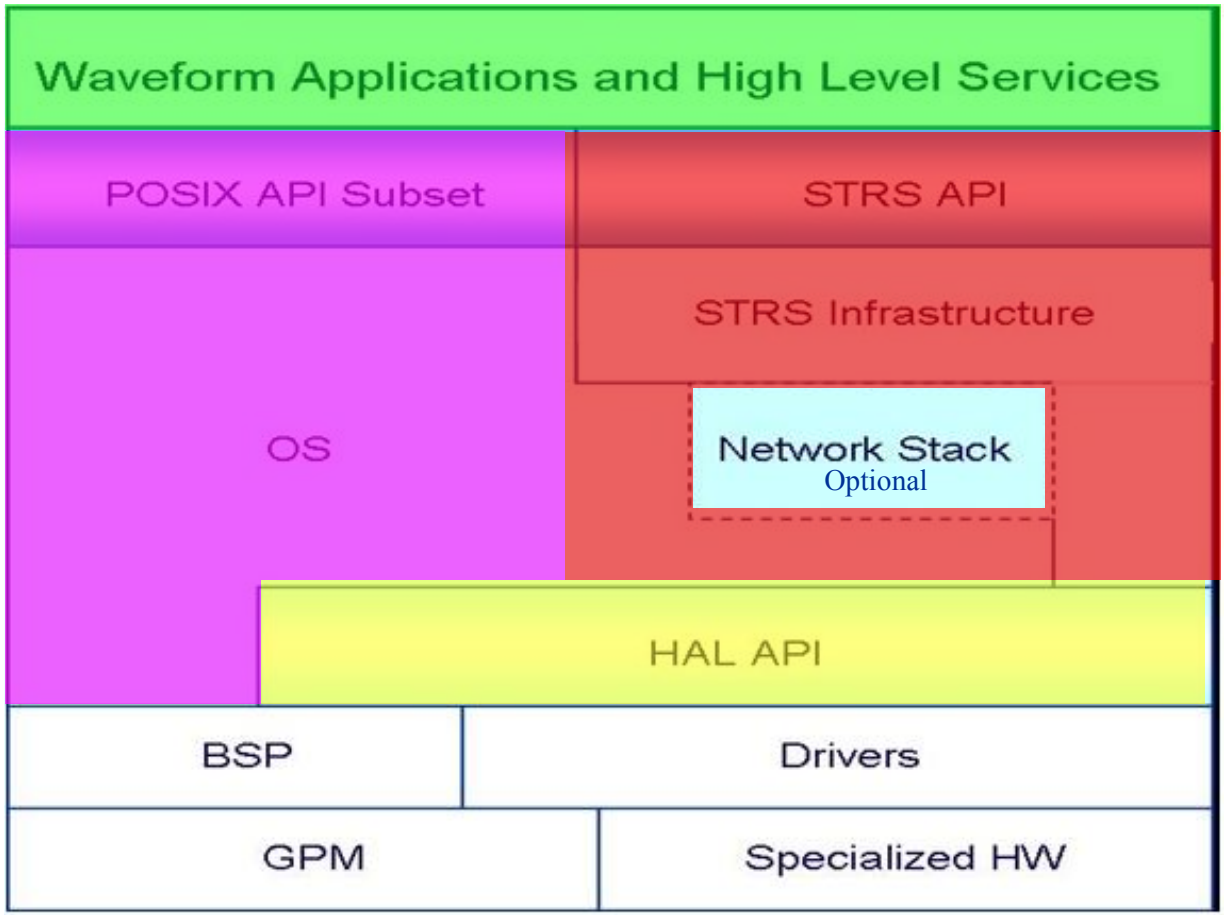
STRS Architecture

- Layer cake model
  - Waveform applications and high level services are insulated from OE by APIs
  - STRS APIs abstract away many platform differences
  - POSIX used to reduce API development
  - OE
  - Hardware Abstraction Layer (HAL)

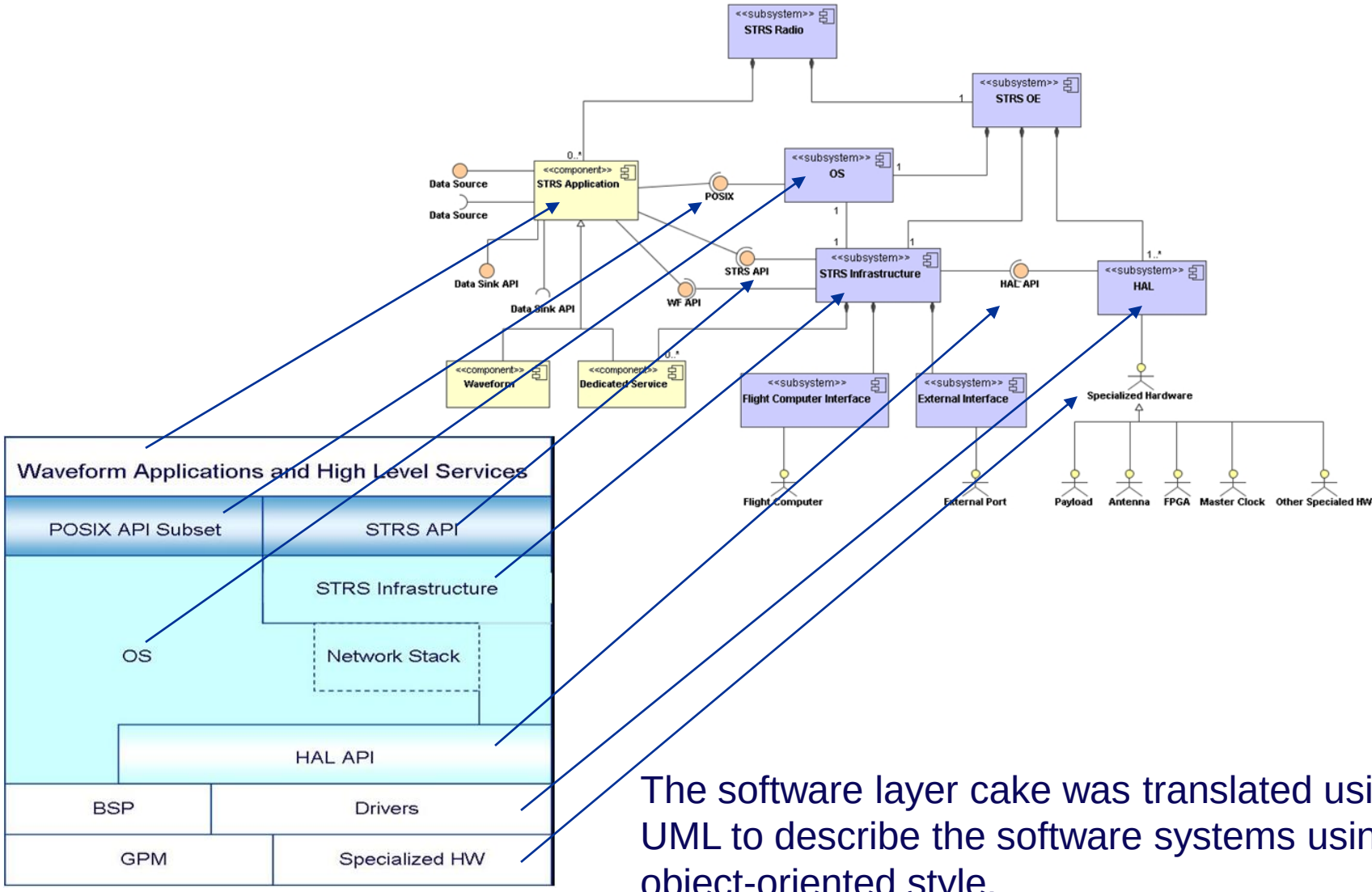


STRS Architecture Conformance

- Conformant to STRS Architecture Standard requirements for applications
- Conformant to STRS Architecture Standard (STRS and WF APIs)
- Compliant to POSIX PSE51 or Subset with Waiver
- Documented HAL and HID as required by STRS Architecture Specification

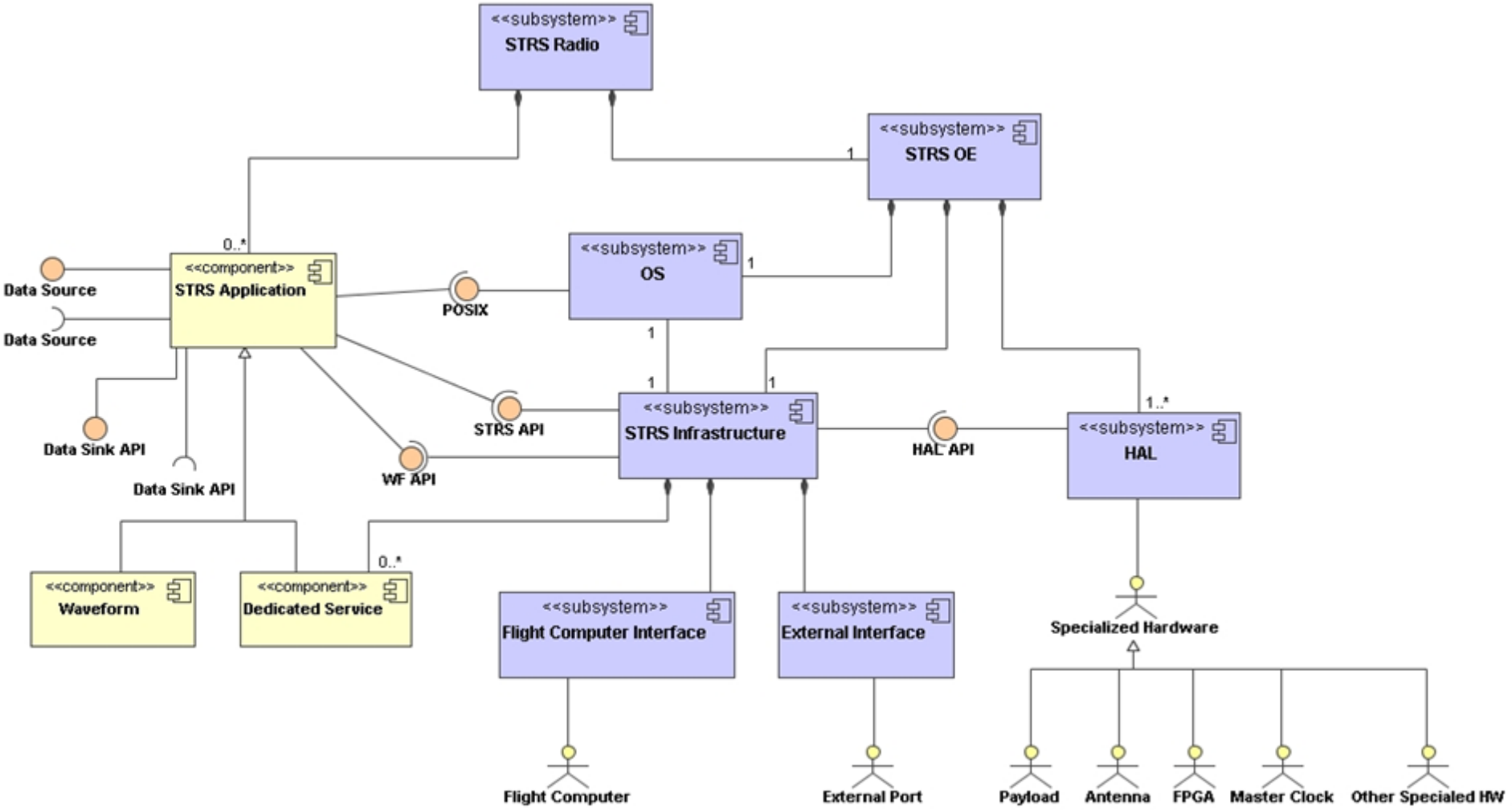


Layer Cake Transition to UML



The software layer cake was translated using UML to describe the software systems using object-oriented style.

STRS Layered Structure



STRS Infrastructure APIs

STRS Infrastructure APIs

- STRS Infrastructure APIs are used:
  - Waveform calls methods in Infrastructure.
  - Infrastructure calls appropriate method in another Waveform, Device, or Infrastructure.
- Purpose:
  - Methods separate a request from the accomplishment of that request.
  - Methods are ‘extern “C”’ so that they can be called from either C or C++.
  - Methods insulate waveforms from having to know how another waveform, device or the infrastructure is implemented.

STRS Infrastructure APIs

Queue Control
- STRS_QueueCreate
- STRS_QueueDelete
- STRS_Read
- STRS_Register
- STRS_Log
- STRS_Write
- STRS_Unregister

Device Control
- STRS_DeviceClose
- STRS_DeviceFlush
- STRS_DeviceLoad
- STRS_DeviceOpen
- STRS_DeviceReset
- STRS_DeviceStart
- STRS_DeviceStop
- STRS_DeviceUnload
- STRS_SetISR

Testing
- STRS_RunTest
- STRS_GroundTest

Attribute
- STRS_Configure
- STRS_Query

Process Errors
- STRS_GetErrorQueue
- STRS_IsOK

Control
- STRS_Initialize
- STRS_ReleaseObject
- STRS_Start
- STRS_Stop

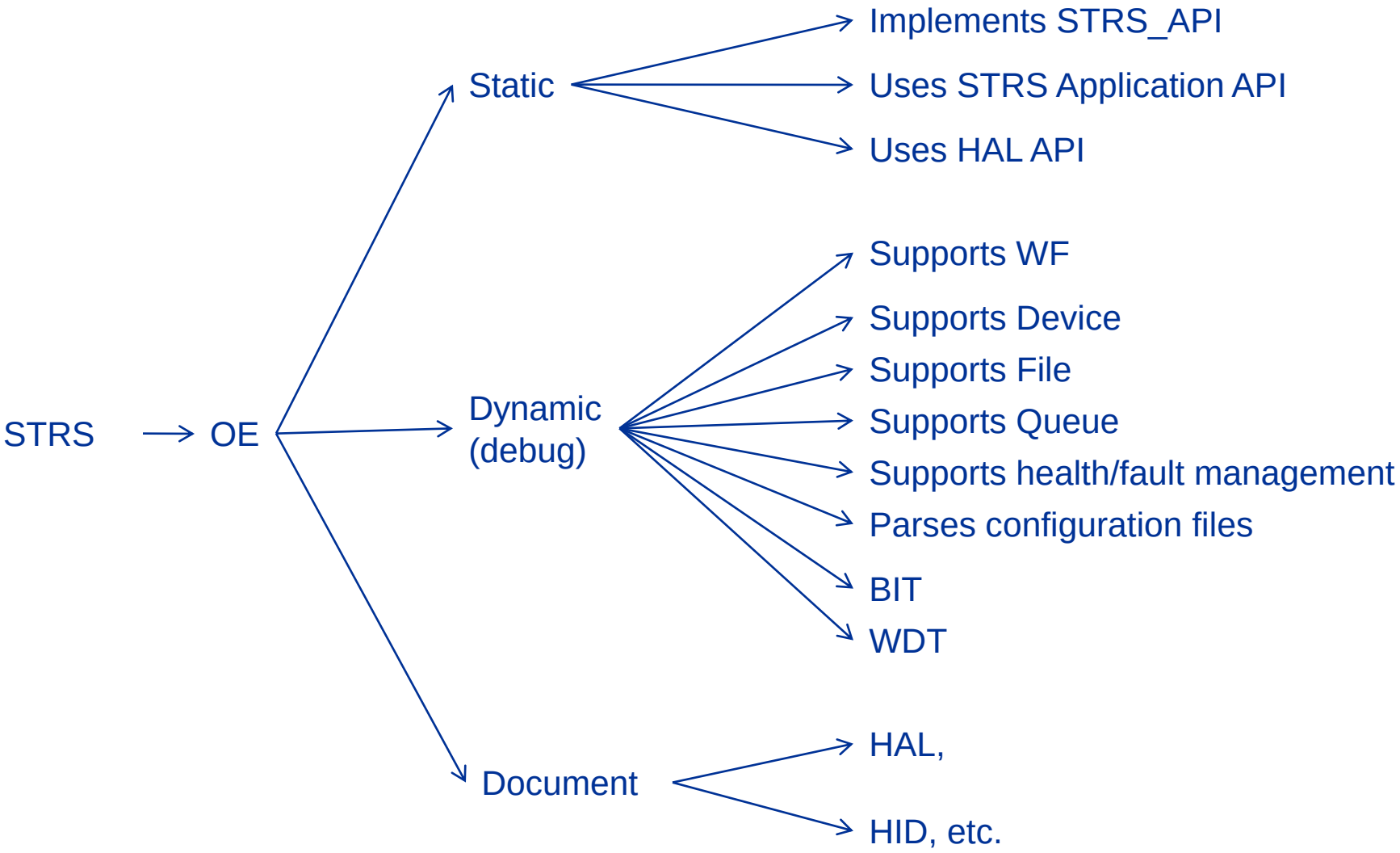
Application
- STRS_HandleRequest
- STRS_InstantiateApp
- STRS_AbortApp

Time
- STRS_GetNanoseconds
- STRS_GetSeconds
- STRS_GetTimeWarp
- STRS_GetTime
- STRS_SetTime
- STRS_Synch

File (Named Area)
- STRS_FileClose
- STRS_FileGetFreeSpace
- STRS_FileGetSize
- STRS_FileOpen
- STRS_FileRemove
- STRS_FileRename

- The STRS Software Architecture presents a consistent set of APIs to allow waveform applications, services, and communication equipment to interoperate in meeting a waveform specification
- These APIs are used in general or to control one waveform from another
- The list to the left is the minimum list of APIs that the STRS platform infrastructure must implement

STRS OE Compliance



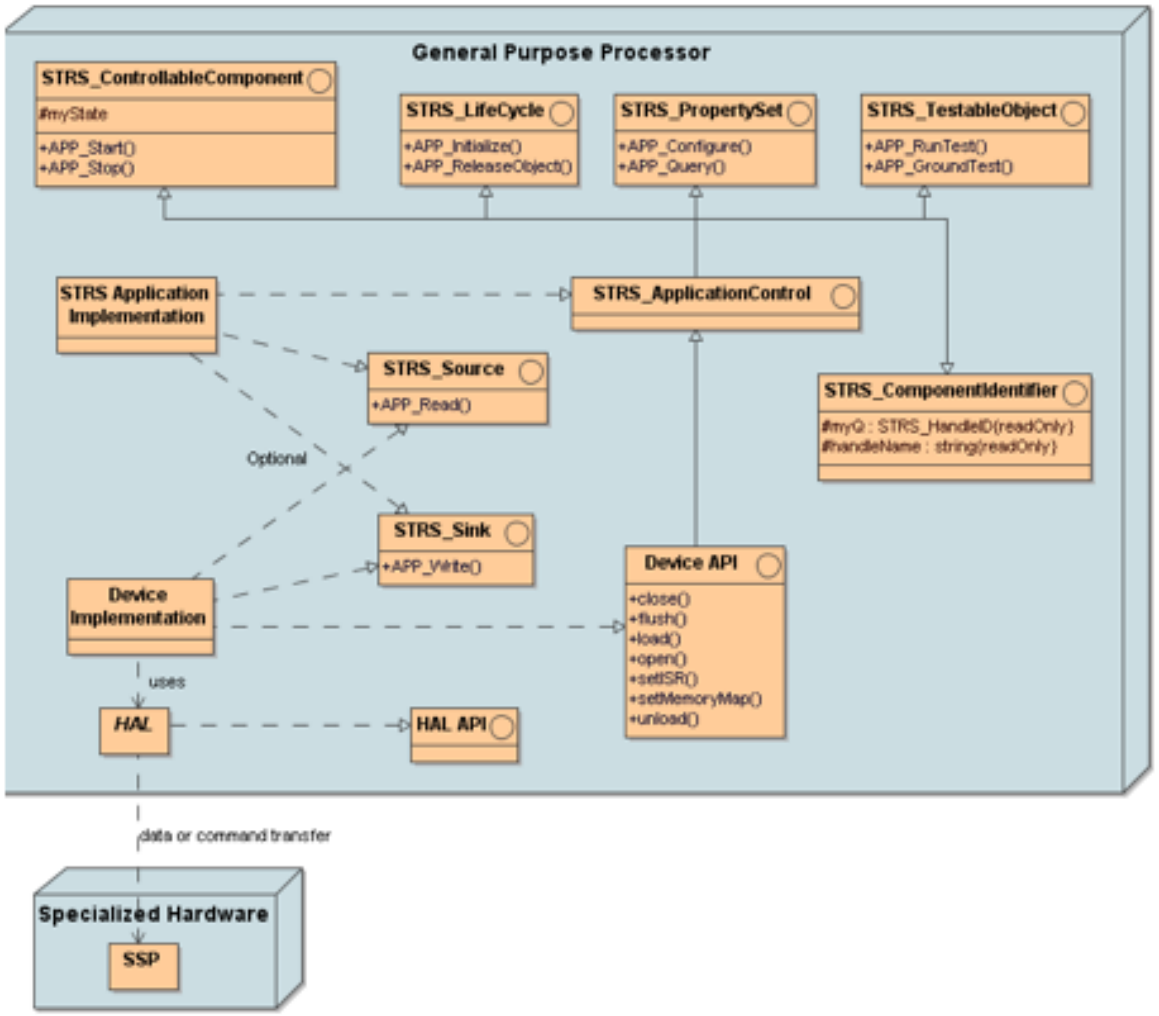
STRS Waveform APIs

STRS Waveform Application Compliance

- A waveform is an STRS Application and must implement the APIs shown in the diagram

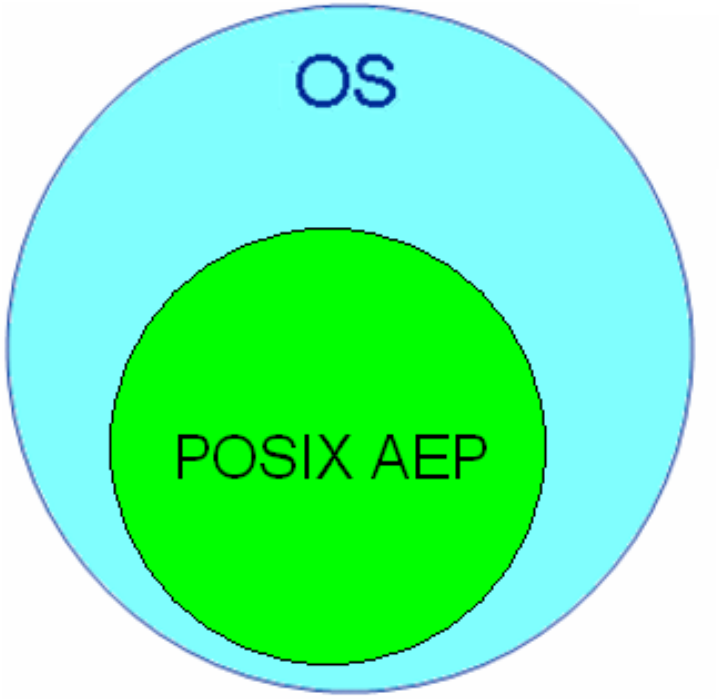
- An STRS Application has OMG similarity; but STRS requires everything, except source and sink (STRS replaces OMG ports with source/sinks)

- The diagram shows how a Device fits in the infrastructure
  - Device is internal, must have the shown functionality
  - Device is an abstraction (proxy) that uses the HAL to get to the hardware
  - No standard for the HAL API. Standard is at Device level (provider)

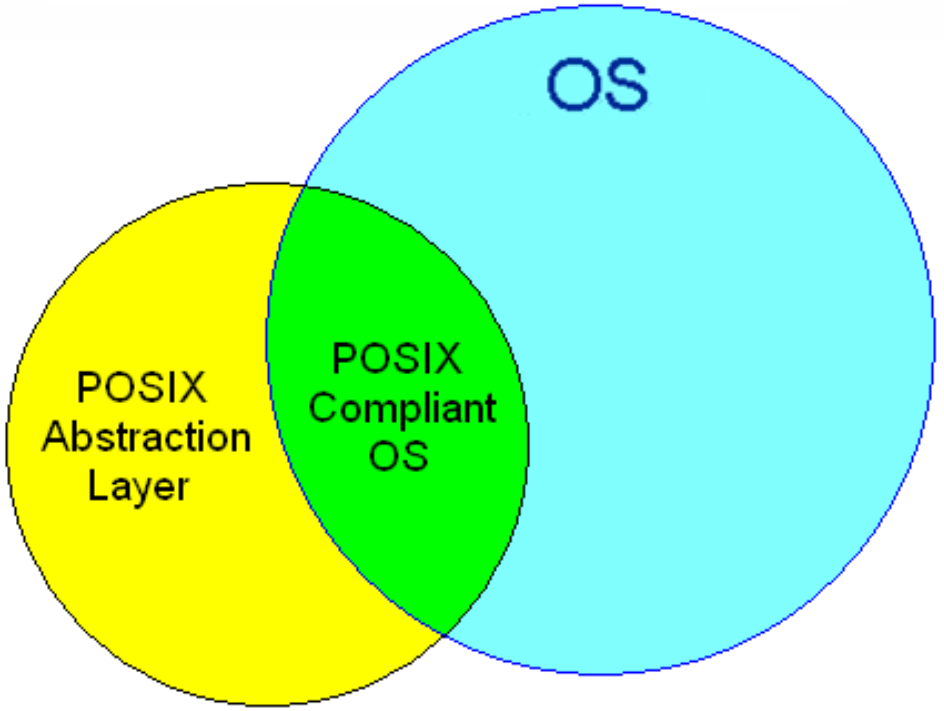


POSIX Compliance/Conformance

POSIX Conformant OS:

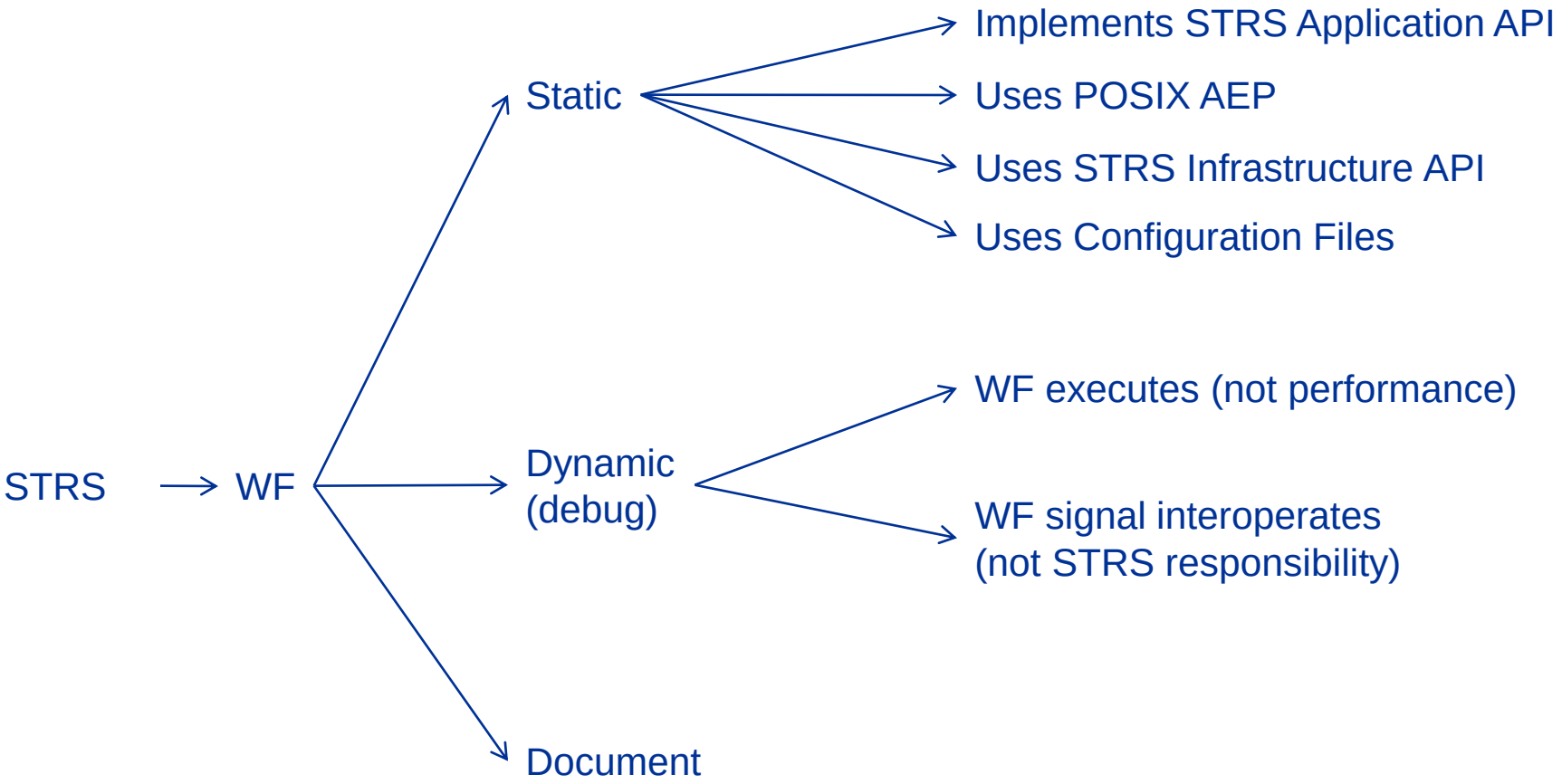


POSIX Compliant OS:



An STRS operating environment can either use an OS that conforms with 1003.13 PSE51 or provide a POSIX abstraction layer that provides missing PSE51 interfaces. For constrained resource platforms, the POSIX requirement is based on waveform requirements so that the **waveforms are upward compatible** (require POSIX methods).

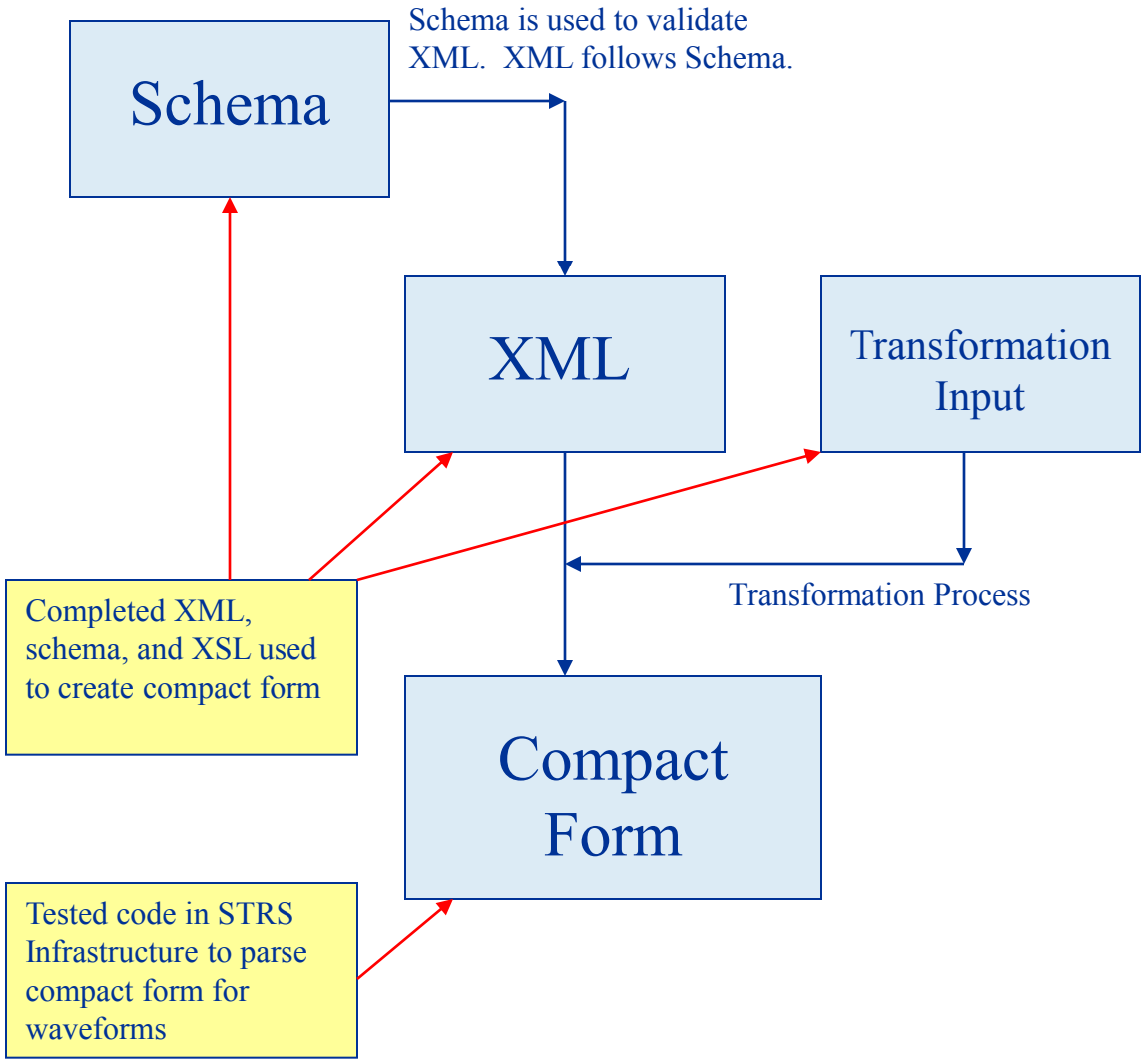
STRS Waveform Compliance



STRS Configuration Files

Configuration Files

- Require schema and XML as part of the architecture
- The required XML should be transformed to a compact format
- The approach for the transformation is not mandated as part of the architecture
- STRS Reference Implementation uses XSL/XSLT to transform XML to an S-expression as compact form



STRS Reference Documents

STRS Reference Documents

- Space Telecommunications Radio System (STRS) Architecture Standard Release 1.02.1, December 2010, NASA TM 2010-216809

http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20110002806_2011001718.pdf

- Space Telecommunications Radio System (STRS) Architecture Goals/Objectives and Level 1 Requirements Document, June 2007, NASA TM 2007-215042.

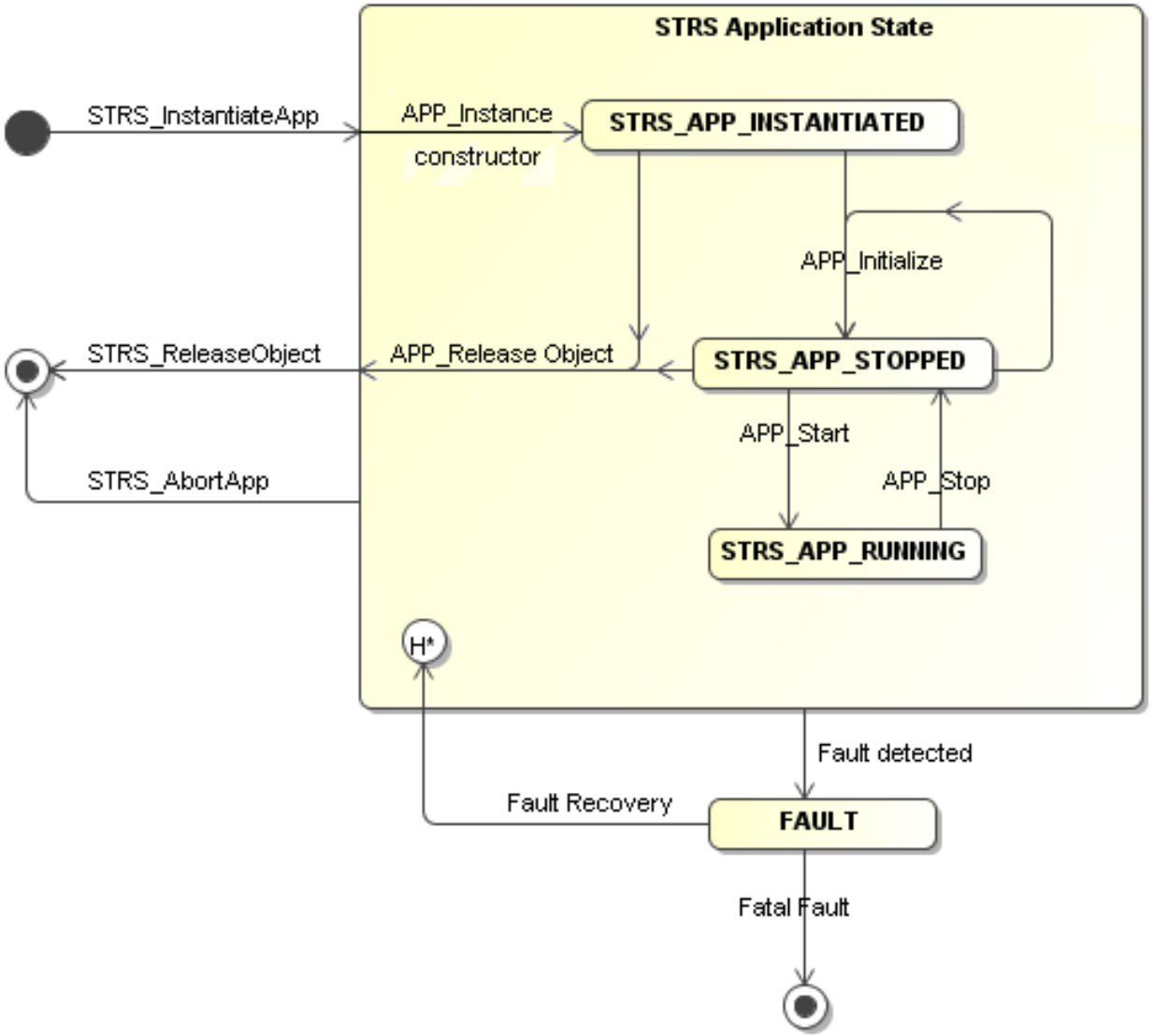
http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080008862_2008008550.pdf

- Space Telecommunications Radio System (STRS) Definitions and Acronyms, May 2008, NASA TM 2008-215445.

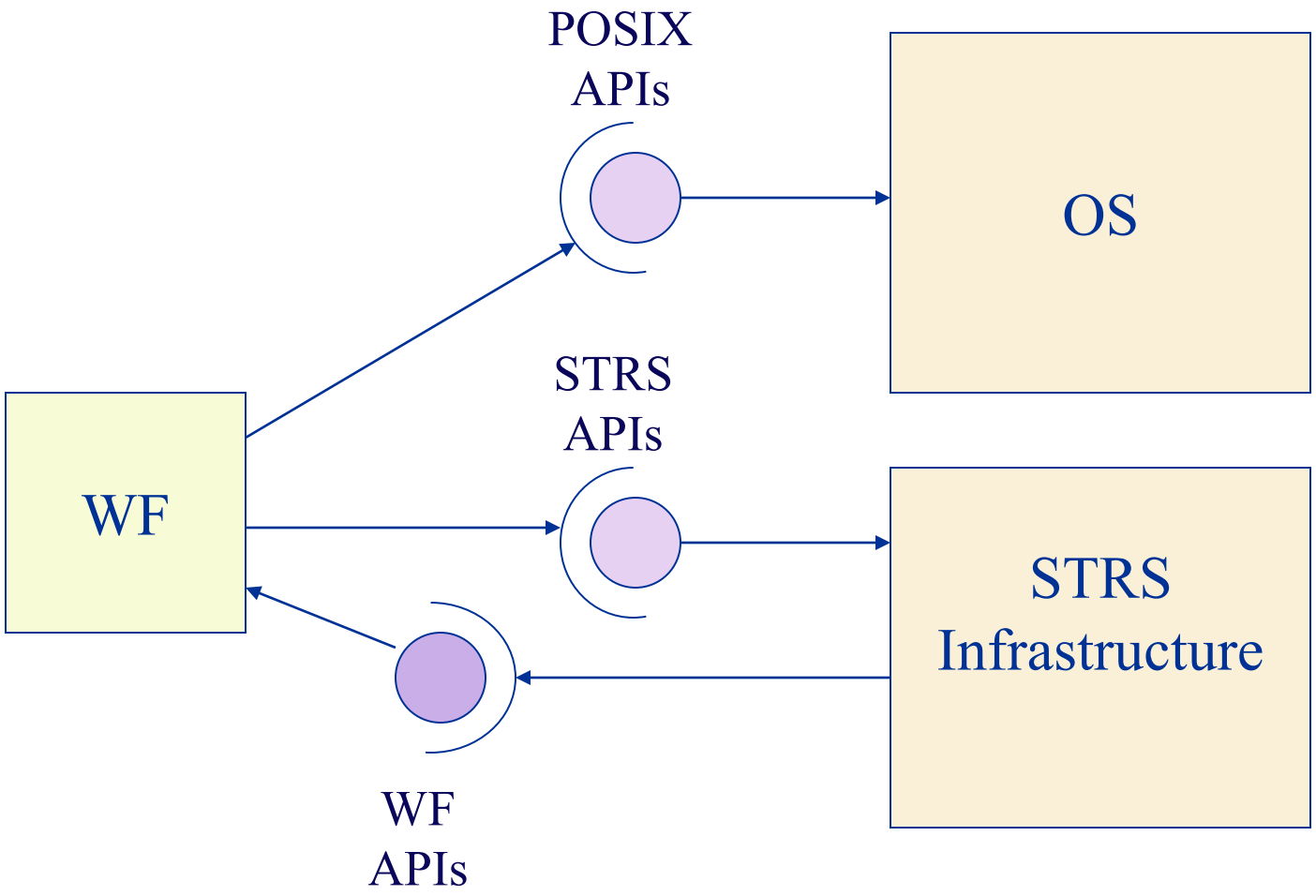
http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090005977_2009004914.pdf

Backup Slides

Waveform State Diagram



Simplified Diagram



STRS Reference Implementation Development Process

Use Case Identifier: Example Use Case

Description: A Description of the use case goes here.

External Actors: External actor(s) involved with this specific use-case.

Related Use Cases: Use cases that related to this use case by interaction or similarity.

Precondition: Any precondition that must exist before this use case can occur. Usually the initial RADIO state and WAVEFORM state is listed.
Triggering event: Event that triggers use case. Not all use cases have triggering events.

Main Flow:
1.This will be an ordered list of steps necessary to perform interaction. This is the nominal flow.
Alternate flows will be listed below.
2.Step 2
3.Step 3
4.Step 4
5.etc.

Resulting event: If completion of use case results in an event it is listed here. Usually the resulting RADIO state and WAVEFORM state is listed.
Post condition: Describe the result of the use case interaction. (This is the post condition from the nominal flow)

Alternatives:
1a) This is where alternate flows are identified. The alternative will be identified by the number of the main flow where the branch occurs followed by a letter a-z.
3a) This is an example of an alternative flow for step 3.
3b) This is the second step in the alternative flow for step 3.
7a-8a) This is an alternative flow for a range of steps from the main flow.

Comments: Comments on use case.

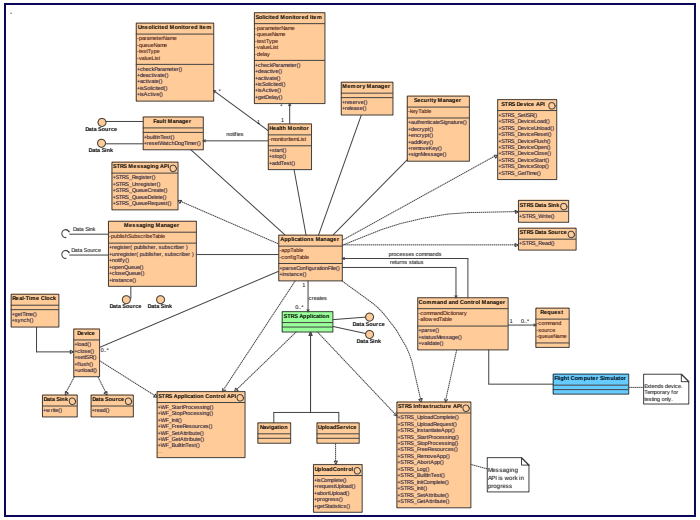
Requirements

Refines Requirements

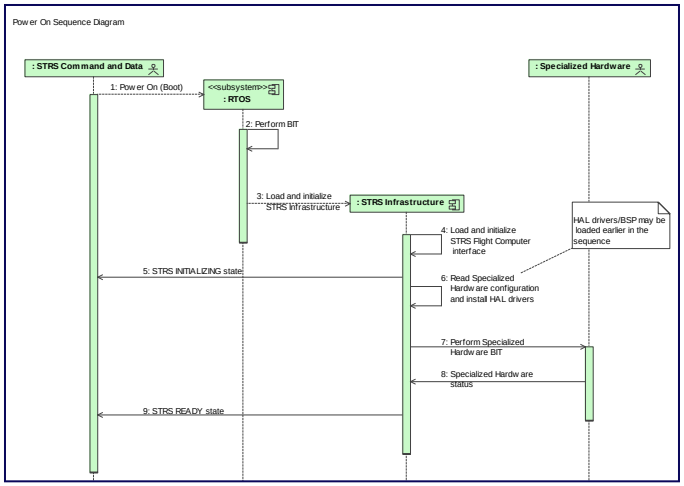
Drives Static View

Drives Dynamic View

Refines Requirements



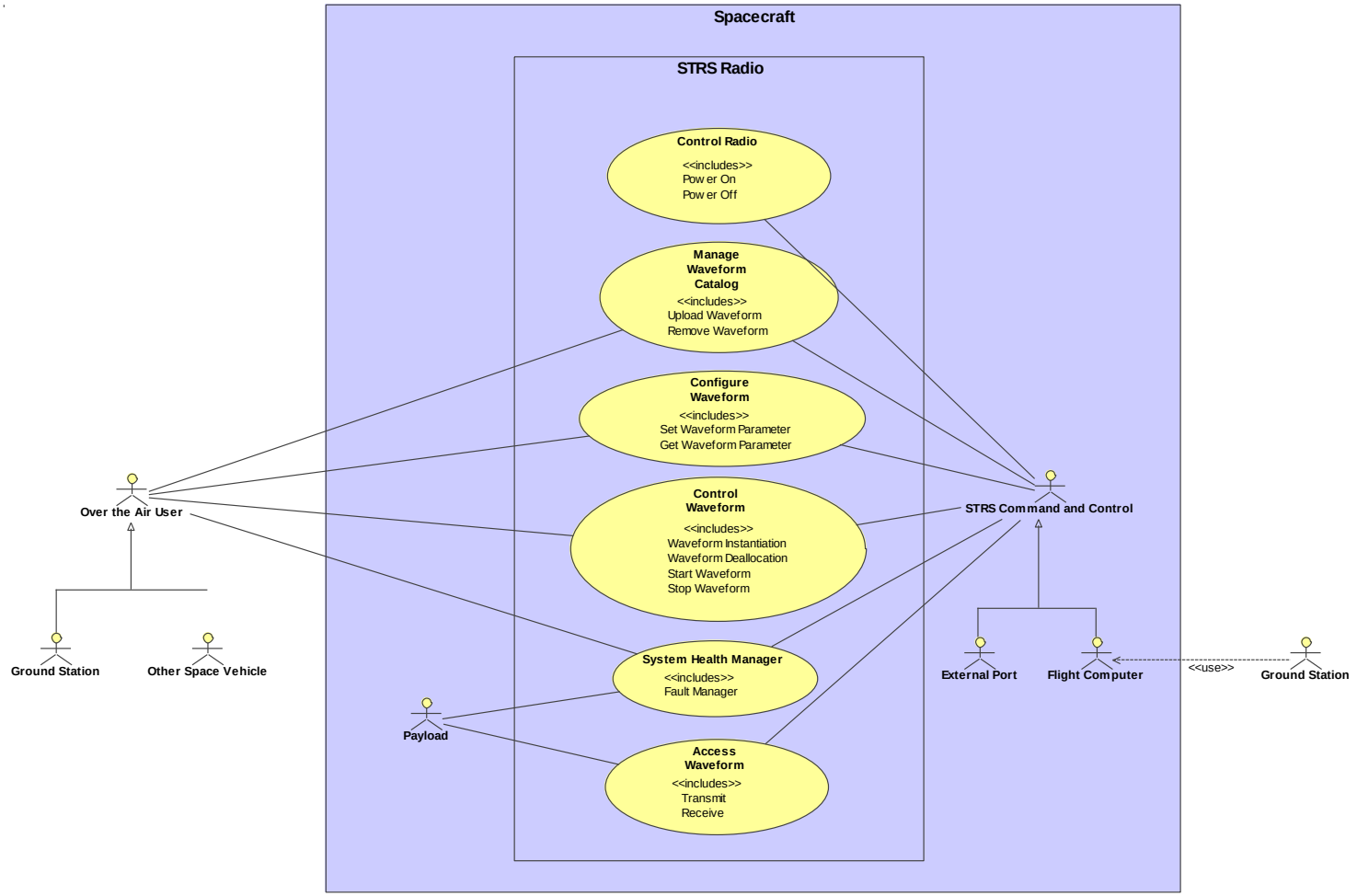
Class Diagram



Sequence Diagram

Use Case

Use Case Overview

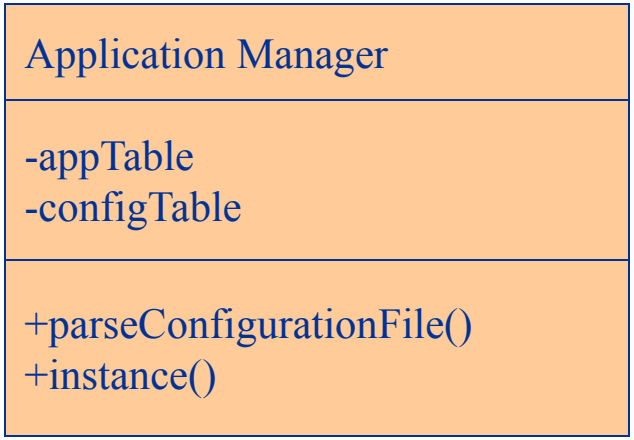


A set of use cases were developed which is a set of scenarios that capture the different ways that external users interact with the STRS radio.

Class Example

Application Manager

- The Application Manager is responsible for the passing of messages or invoking commands in other application objects such as devices, waveforms, or services actively running on the STRS radio.
- It is responsible for creating or aborting application objects, waveforms, or services.
- It is also responsible for parsing the Configuration Files and setting corresponding values in the appropriate classes.



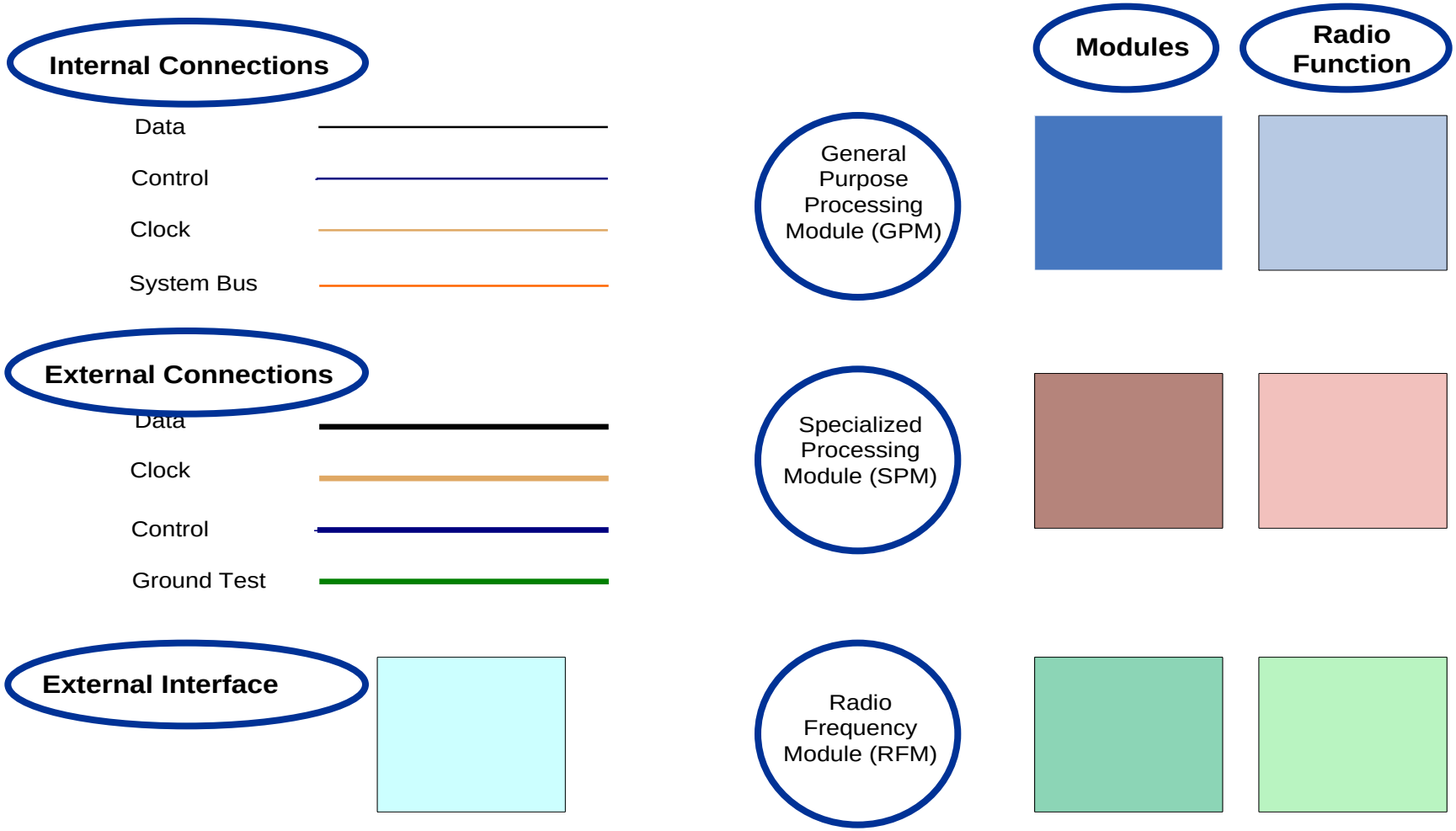
Above is an example of the UML representation of a Class

Name – *Name that identifies the class and describes the functionality*

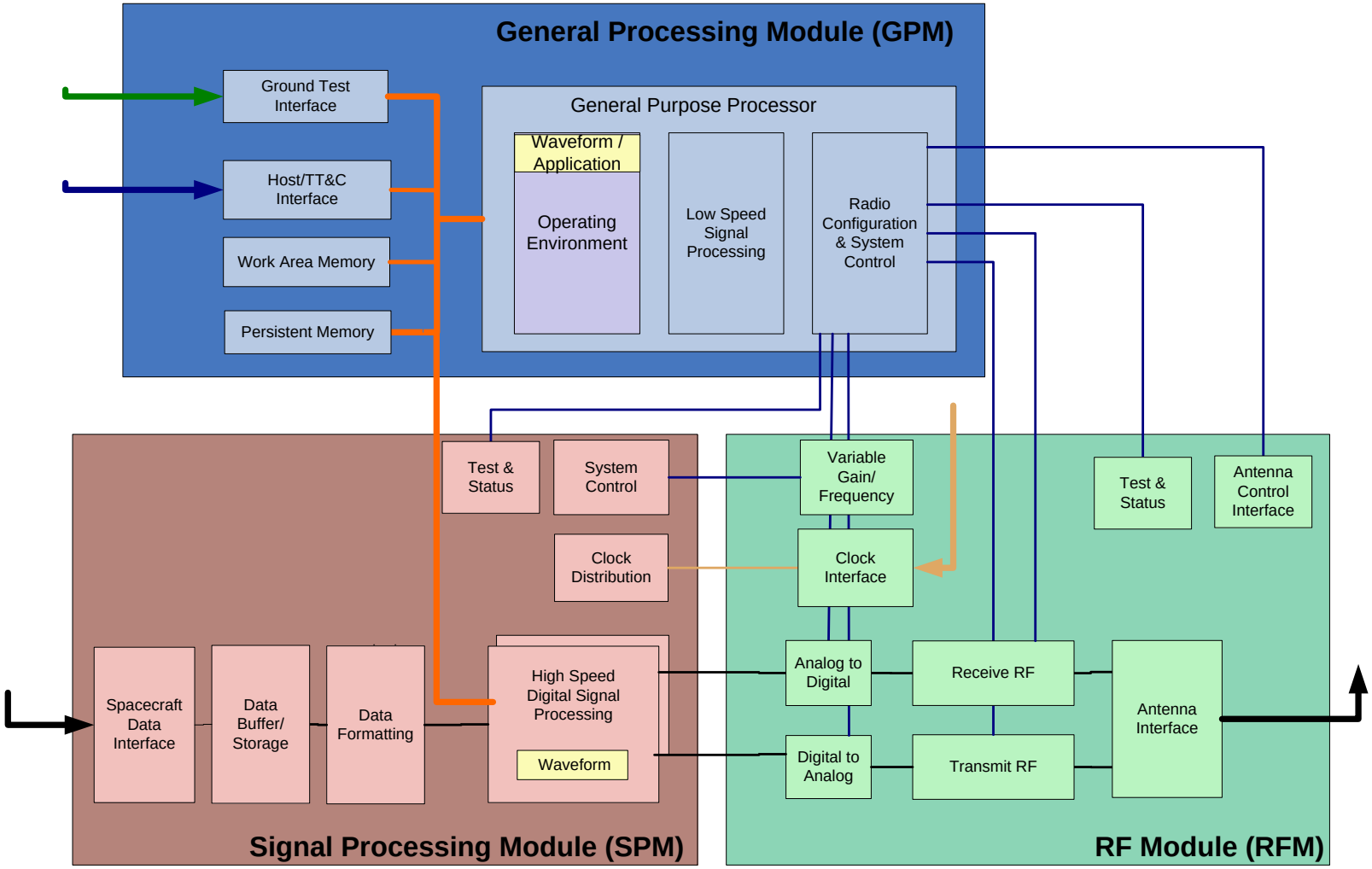
Attributes – *Variables containing the applicable data*

Methods – *Functions that are called to implement some operation*

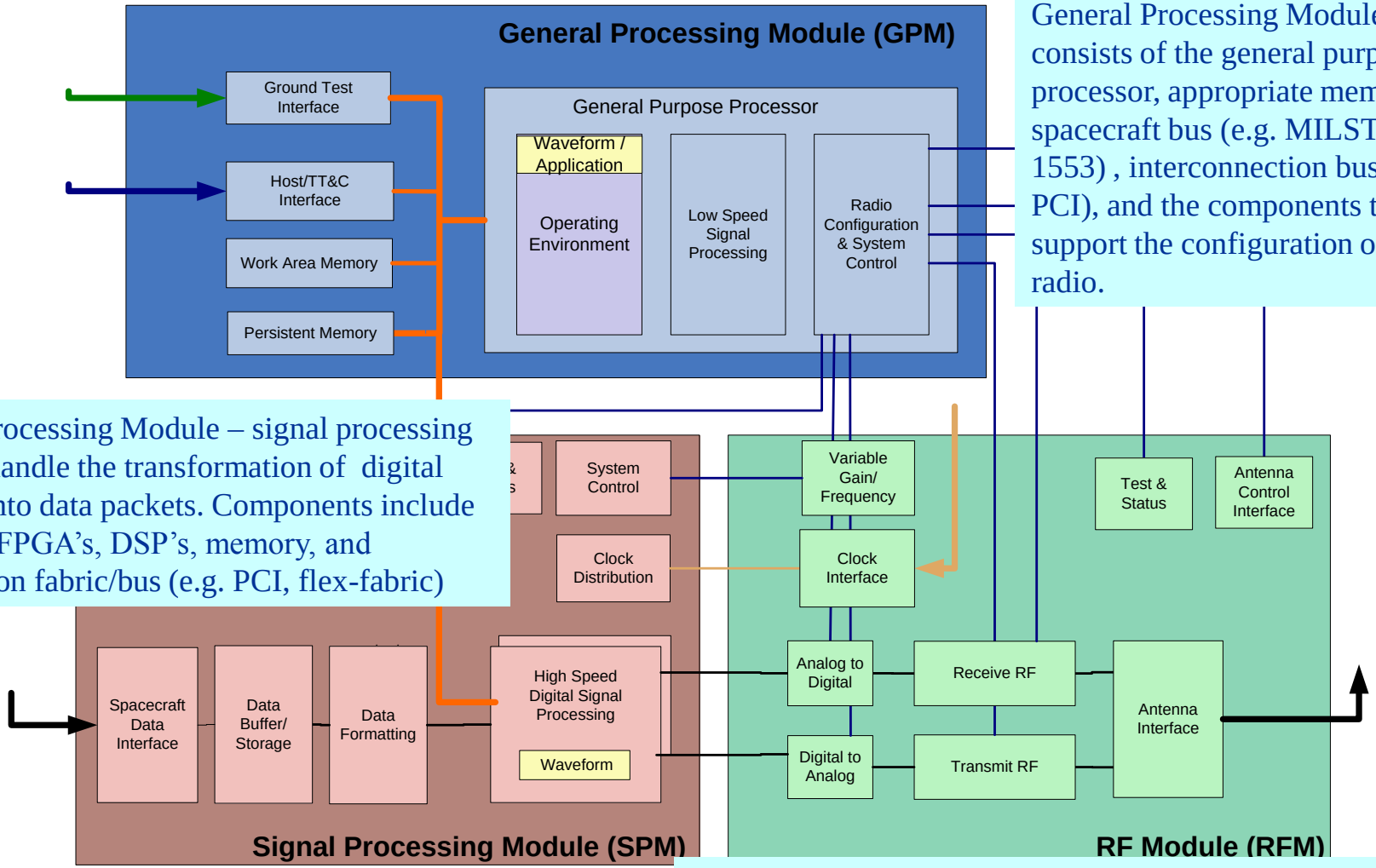
STRS Open Architecture Hardware Representation



SDR/STRS Hardware Functional Diagram



SDR/STRS Hardware Functional Diagram



General Processing Module – consists of the general purpose processor, appropriate memory, spacecraft bus (e.g. MILSTD-1553) , interconnection bus (e.g. PCI), and the components to support the configuration of the radio.

Signal Processing Module – signal processing used to handle the transformation of digital signals into data packets. Components include ASIC's, FPGA's, DSP's, memory, and connection fabric/bus (e.g. PCI, flex-fabric)

RF Module –handles the RF functionality to transmits/receive the digital signal. Its associated components include RF switches, diplexer, filters, LNAs and power amplifiers.

STRS Hardware Functional Diagram

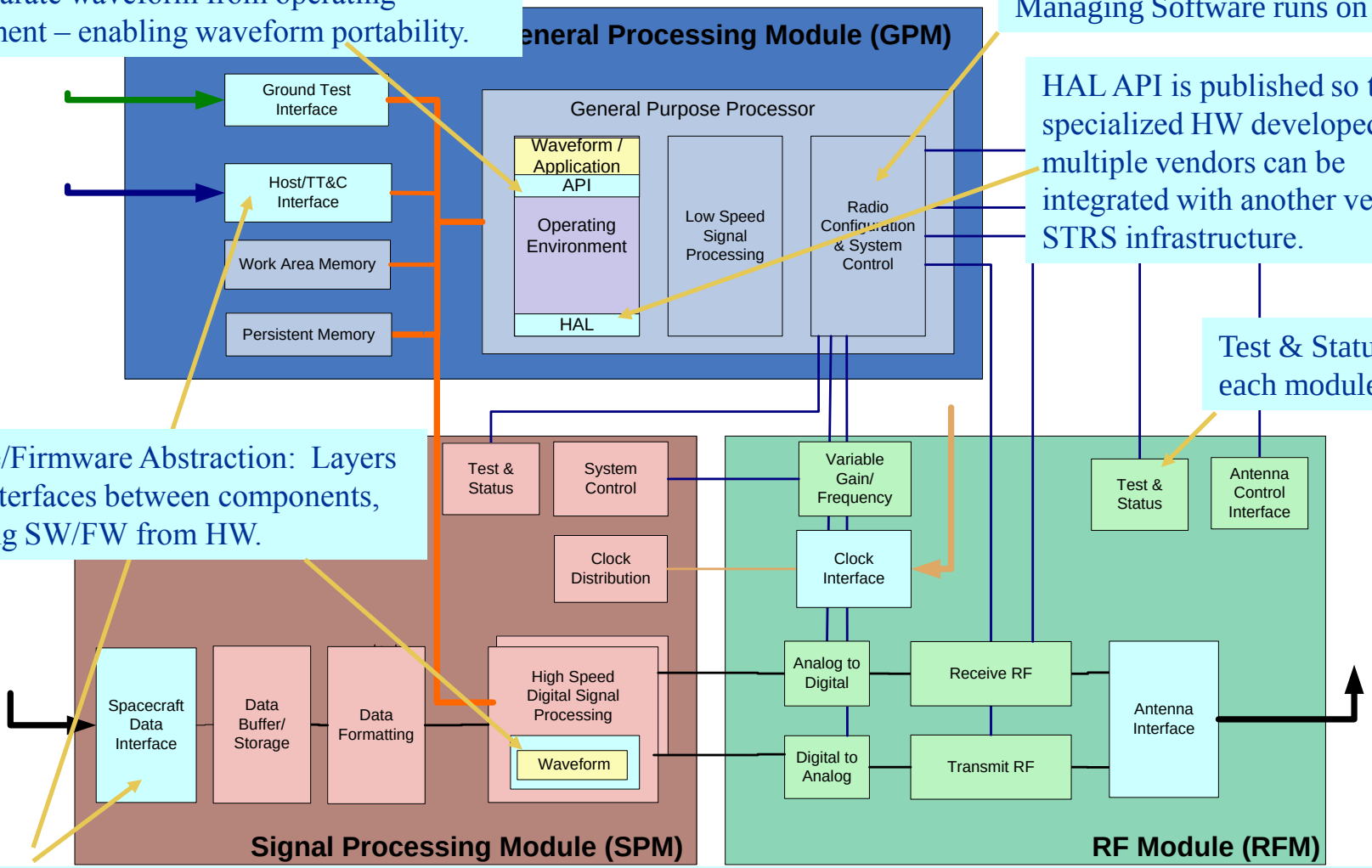
APIs separate waveform from operating environment – enabling waveform portability.

Managing Software runs on GPM

HAL API is published so that specialized HW developed by multiple vendors can be integrated with another vendor's STRS infrastructure.

Test & Status on each module

Software/Firmware Abstraction: Layers define interfaces between components, separating SW/FW from HW.



Module Interfaces abstract and define the module functionality for data flow to waveform components. Enables multiple vendors to provide different modules or add modules to existing radios. Electrical interfaces, connector requirements, and physical requirements are specified/published by the platform provider.

The End