

# Design Patterns of the JTRS Infrastructure

Donald R. Stephens, Cinly Magsombol, and Chalena Jimenez  
 JPEO JTRS San Diego, CA

## ABSTRACT

The concept of the Joint Tactical Radio System (JTRS) infrastructure is to define a radio host environment for the execution of waveforms and applications. Software developers are guaranteed a specific set of real time operating functions, distributed messaging through Common Object Request Broker Architecture (CORBA), and radio domain-specific interfaces such as Global Positioning System (GPS), Ethernet, audio, etc. This promotes reuse and portability of waveform components. The hardware dependencies are isolated from the applications by the Application Program Interfaces (APIs) defined for radio devices.

Software design patterns have been developed by the JTRS community to define a scaleable and extensible infrastructure. Aggregation, least privilege, extension, explicit enumeration, and deprecation enable the infrastructure to support varying missions and form factors. These design patterns permit the instantiation of a radio infrastructure suitable for the platform footprint and resources of a single-channel handheld radio, yet also permit it to be scaleable and extensible to the requirements of a multiple-channel wireless networking gateway.

## INTRODUCTION

The principle of the JTRS infrastructure is to provide a host environment supported by all radio sets within the JTRS product line [1]. Common interfaces and minimum radio services are defined by the infrastructure for all JTR Sets. This enables a JTRS waveform or application developer to generate software products that can execute upon any radio in the product line.

Definition of the JTRS infrastructure has been a collaborative product of the JTRS Interface Control Working Group (ICWG). The primary emphasis in standardization has been upon the waveform-to-set interfaces illustrated in Figure 1. Only the interfaces between the waveform and the radio are standardized. The internal interfaces and transport mechanisms of the radio are defined as required by the radio provider. The intent is to provide portability or reuse of the waveform between radio platforms and not necessarily reuse of the radio operating environment software.

Figure 2 illustrates the definition and deployment of the JTRS infrastructure [2]. Regardless of the mission or size of the JTRS radio, the services and interfaces are

supported across the product line. Portability and compatibility for General Purpose Processors (GPP) are obtained by compliance to the Software Communications Architecture (SCA) [3] which includes an AEP specifying a POSIX Real Time Operating System (RTOS) subset and CORBA middleware.

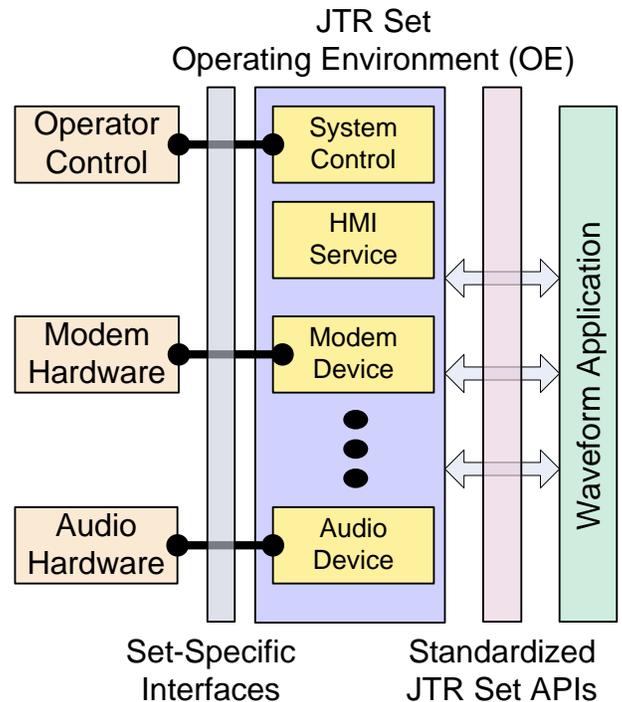


Figure 1 JTR Set and Waveform Interfaces

The JTRS infrastructure promotes portability of FPGA and DSP-based software by the definition of the Modem Hardware Abstraction Layer (MHAL). This specification enables communication between software components distributed on different non-CORBA enabled processing elements.

The primary goals of the infrastructure are: a) portability, b) scalability, c) extensibility, and d) backward-compatibility. To achieve these goals, the ICWG has implemented several design patterns and strategies for the JTRS Infrastructure

Over 3.5M source lines of code have been generated for the JTRS program and the definition of the JTRS infrastructure maintains the viability and applicability of this software for future radio deployment. The design patterns of aggregation, least privilege, extension, explicit enumeration, and deprecation are described below.

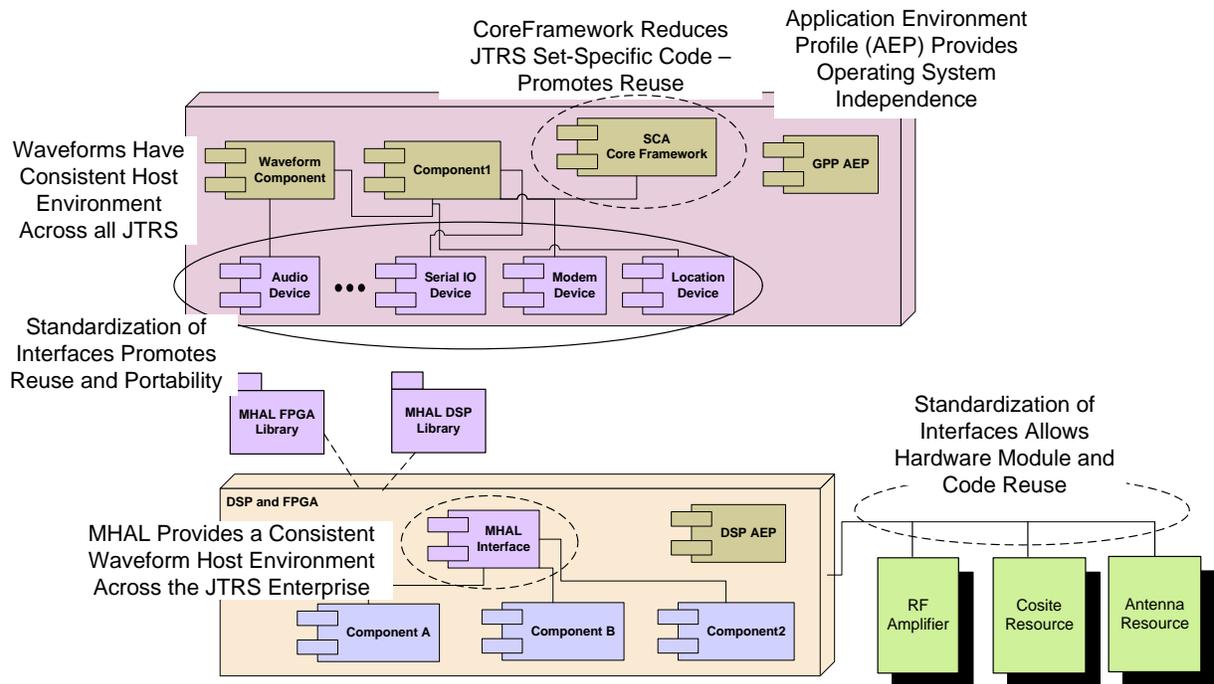


Figure 2 Definition of the JTRS Infrastructure

### Aggregation

The JTRS infrastructure is designed to support all JTRS sets and missions without requiring the smaller radios to provide unnecessary services and devices. The design pattern illustrated in Figure 3 has an API that consists of two separable interfaces. With the aggregation design strategy, radio set developers are permitted to overload provided ports by aggregating as many interfaces as desired into a single provided port. This strategy can reduce the number of CORBA servants within the set which consequently reduces the memory resources required by the implementation.

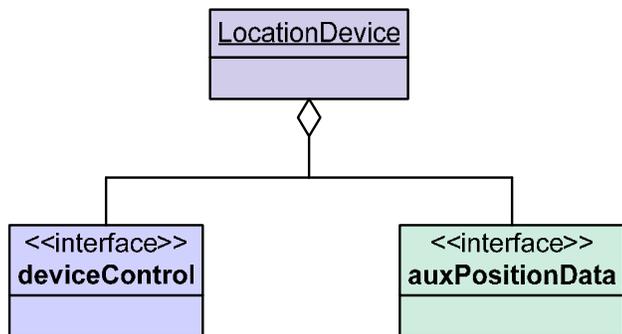


Figure 3 Interfaces are Aggregated for Components

Actual port names for the components are not specified in the APIs; only reference names are used for

documentation purposes. The actual port names in the JTRS set are intended to be specified by the set provider.

In Figure 3, the location device is required to provide two separate interfaces, *deviceControl* and *auxPositionData*. The interfaces are not coupled together, which either allows separation or unification, depending upon the implementation of the radio set's operating environment.

The strategy is illustrated through Figure 4. In the SCA, ports are defined as connection points between software components. These connection points allow components to be distributed anywhere within the radio. The SCA Core Framework's application factory connects the two components by exchanging the addresses of peer ports at runtime. Generally a port is associated with a specific set of CORBA interfaces, but the JTRS infrastructure permits either aggregation or separation of the ports as shown in Figure 4.

### Least Privilege

A related design pattern is illustrated in Figure 5. In the example, the MailService API does not expose the base SCA interfaces to a waveform component, or any other unnecessary interfaces. The rationale is that it is undesirable for a waveform to have the capability to start or stop a radio set hardware component or service. Such control could subsequently affect other waveforms or radio channels.

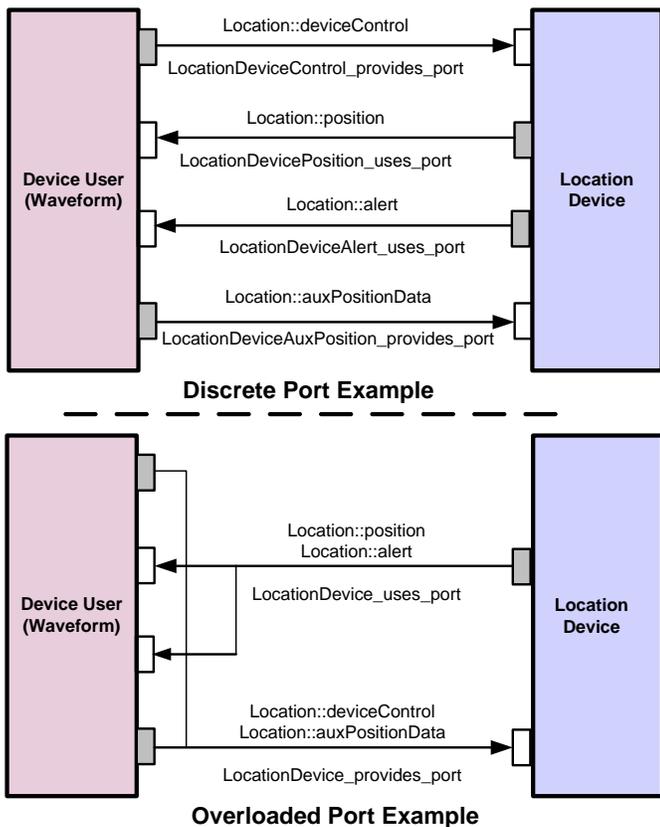


Figure 4 Aggregation of SCA Ports

**Error! Objects cannot be created from editing field codes.**

Figure 5 Principle of Least Privilege for Interface Definition

Control of the JTR set by the waveform or application should be limited to strictly those operations needed for configuration and control of the waveform. Because the base SCA interface is not available to the waveform, it cannot invoke operations that could affect the state of the JTR set or other waveforms. This principle of least privilege – exposing only the essential interfaces required by a client component is a guideline for all JTRS APIs.

### Extension

The extension design pattern is depicted in Figure 6. The concept is to define a base API and optional extensions or services that could be included in a radio set as required by mission or application. As an example, a service such as the MailService is defined in Figure 6. It consists of a base interface which would be provided by every JTR set that is required to provide the service. In addition, the MailService API could be extended for sets that required a POP3 interface for

retrieving email from a remote server. As shown in Figure 6, this additional interface would be optional and perhaps unnecessary on small form factor or hand-held sets. For missions or sets that did require this feature, the behavior and interfaces would be standardized for the JTRS enterprise.

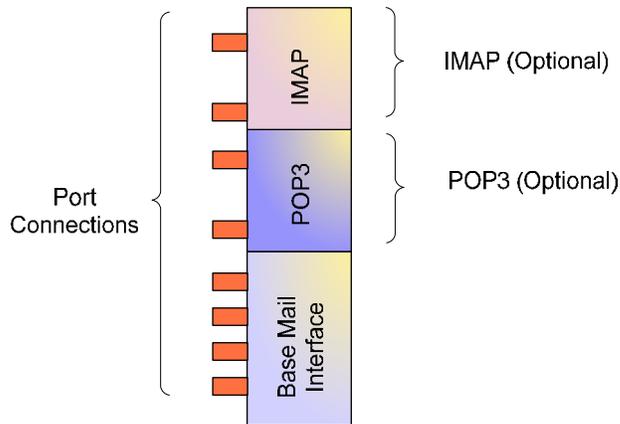


Figure 6 Base APIs can be Extended

Figure 7 lists the CORBA Interface Definition Language (IDL) for the example’s base mail service interface. Note that the IDL keyword, “module”, is used to define the namespace “MailService”. As discussed previously, interfaces such as MailBox, are later aggregated inside the “MailService” namespace. To invoke a method, it is necessary to use both the module name and specific interface. As an example, to determine whether the mailbox was empty, MailService::MailBox::isEmpty() would be invoked.

```

module MailService{
    typedef unsigned short      ExtEnum;
    typedef sequence<ExtEnum>
                               ExtEnumSequence;

    const      ExtEnum MAIL_BASE = 0;

    interface MailBox{
        Boolean isEmpty();
        void clearBox();
        unsigned short numberMessages();
    };
};

```

Figure 7 IDL for the Base Mail Service (mailBase.idl)

To minimize the processing and memory resource requirements for a JTR Set, the JTRS Infrastructure

would not require every radio to implement the POP3 protocol. However, if the radio does need to support POP3, then the JTRS standardized POP3 extension API is implemented on the set. Example IDL is illustrated in Figure 8 where a new interface, Pop3Box, is defined for the additional capability.

```
#include "mailBase.idl"
module MailService{
    const ExtEnum MAIL_POP = MAIL_BASE + 1;

    interface Pop3Box{
        long    address();
    };
};
```

Figure 8 IDL for the Mail Service Extension

As with the base interface, the MailService namespace is used to encapsulate the interface Pop3Box. Since this is the same module name as the base interface for the MailService, the IDL in Figure 8 aggregates the Pop3Box interface with the base interface. Identical to namespaces in C++, modules in IDL can be reopened. This allows the IDL compiler to process Figure 7 first, and later discover that Figure 8 provides additional code for the same module. To obtain the POP3 address, the method MailService::Pop3Box::address() would be invoked.

The advantage of using aggregation instead of inheritance for JTRS Standards is illustrated in Figure 9. Fictitious development dates are assigned to the software builds to provide a time reference. Waveform A is developed using the base mail service interface and the POP3 extension. Similarly in time, JTR Set 1 is developed with the same set of APIs. Later in time, JTR Set 2 is developed with an additional MailService extension. However, Waveform B was developed with only the IMAP mail protocol – the POP3 protocol was not needed in its implementation.

The compatibility matrix at the bottom of Figure 9 illustrates that Waveform A does not need to be refactored to execute on JTR Set 2. Aggregation allows software generated with the JTR Standards APIs to be backward compatible with previous implementations. As would be expected, in order for JTR Set 1 to support Waveform B, new mail service protocol capability must first be added to the set. Then it could also host the new waveform.

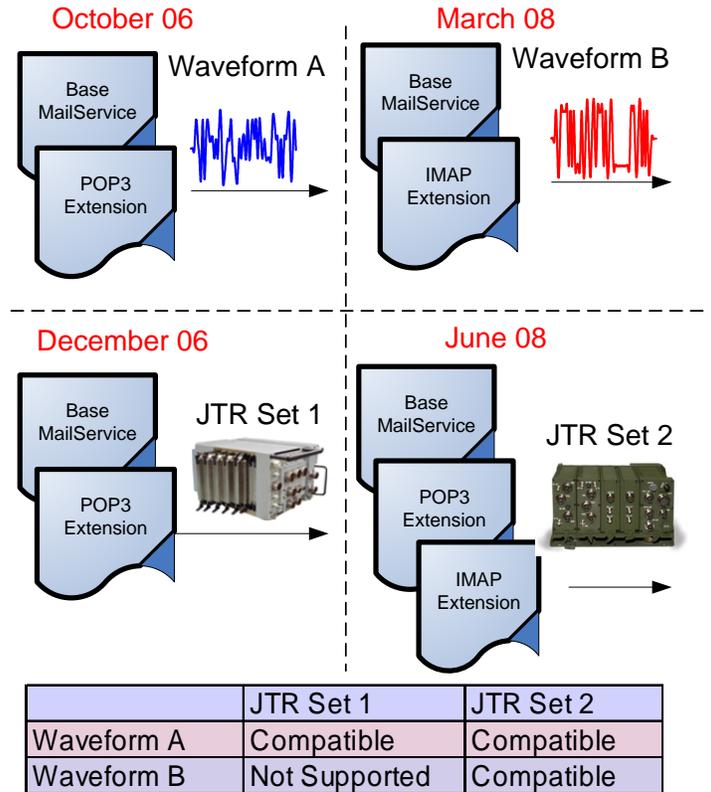


Figure 9 Benefit of Interface Aggregation

### Explicit Enumeration

During the development of the JTRS Standards APIs, enumerations in APIs were identified as particularly non-extensible. An example of an enumeration is illustrated in Figure 10. If it is desired to add a Local Mail Transfer Protocol (LMTP) algorithm to this list, the interface must be redefined which subsequently requires all of the existing software to be updated whenever new capability is added such as the new protocol.

```
enum MailTypes{
    MAIL_BASE,
    MAIL_POP,
    MAIL_IMAP
};
```

Figure 10 Example Implicit Enumeration

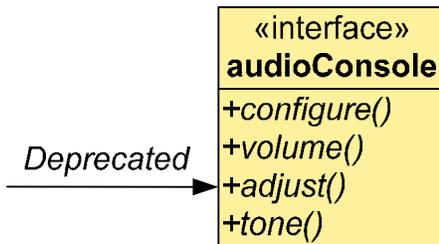
To mitigate the difficulty of extending enumerations, the design strategy adopted by JTRS Standards is illustrated in Figure 11. Instead of directly using enumerations, the constant MAIL\_POP is defined only in its extension. This avoids the difficulty of continually updating the code base as new capability is added to the APIs.

```
#include "mailBase.idl"
module MailService{
    const ExtEnum MAIL_POP = MAIL_BASE + 1;
};
```

Figure 11 Explicit Enumeration Through Constants

### Deprecation

During JTRS maturation and fielding, APIs must evolve or become obsolete. Deprecation of methods and interfaces has been selected by the JTRS ICWG to gracefully accommodate obsolescence. Figure 12 illustrates an example interface for an audioConsole. Originally *adjust()* was a method to control the volume of the audio stream. Small form factor radio developers objected to an additional function call because *volume()* could perform the same functionality. In this example, the dilemma for the JTRS enterprise would be that software already deployed in radios required the *adjust()* method. To avoid an immediate refactoring of all the code in the JTRS product line, the *adjust()* method would be deprecated as illustrated in Figure 12. This informs waveform developers generating new code that the *adjust()* method should not be used for future development. JTR set developers are similarly alerted that older waveforms ported to the set may require that method to be supported by the JTR set.



### SUMMARY

The JTRS program has defined a radio infrastructure tailored for DoD communications. The JTRS infrastructure specifies not only minimum capability and services available for any JTRS radio, but also specifies the interfaces and services for enhanced feature sets. The scalability and extensibility of the infrastructure is achieved through the application of design patterns.

Additionally the design patterns restrict the scope of control for waveforms and applications, limiting their ability to affect other waveforms and applications executing upon the radio set. The design patterns also allow for future evolution and refinement of radio services and interfaces. Backward compatibility with previous waveforms and applications is provided through the definition of the design patterns.

### References

- [1] Stephens, D.R., Salisbury, B., Richardson, K., “JTRS Infrastructure Architecture and Standards”, MILCOM 2006.
- [2] Schivonne, L., North, R.C., Browne, N., Joint “Tactical Radio System – Bringing the GIG to the Tactical Edge”, MILCOM 2006.
- [3] Hayes, N., “The JTRS SCA specification... the past, the present, and the future...”, MILCOM 2005.