Ada in an On-Board Military Communication System

Victor D. Albertini
L-3 Communications
640 North 2200 West
Salt Lake City, UT 84116
1-801-594-2909
victor@csw.L-3com.com

Craig J. Berrett
Iomega
1821 West Iomega Way
Roy, UT 84067
1-801-778-3487
berrettc@iomega.com

1. ABSTRACT

This experience report was drawn from work performed on a real-time communication system for the military. The hardware components of this system are modular, which facilitates building systems with differing personalities based upon the assembled components. Consequently, the primary goal of the software design was to achieve a modular software design for command and control of the system. The developers feel that Ada 83 was effectively utilized to model the hardware of this sophisticated communication system.

1.1 Keywords

Ada, Modular Software, Hardware to Software Mapping

2. INTRODUCTION

In this report, a brief description of the application is provided as background for the reader. Next the basic software design of the application, which provides a one to one mapping between the actual hardware components of this communication system and the Ada packages and data structures developed to model the hardware is illustrated. Finally, the utility of this design approach for system modification or enhancement is discussed.

3. The Application

This military application consists of an airborne relay platform with hardware components to support from one to three simultaneous data links; i.e., air to air, air to ground, and queuing. Figure 1 is provided to illustrate the communication links for the relay platform. From the perspective of a relay, the air to air communication link is referred to as a High Rate Receive Link, the air to ground link is referred to as a High Rate Transmit Link, and the queuing link is called a Low Rate Receive Transmit Link.

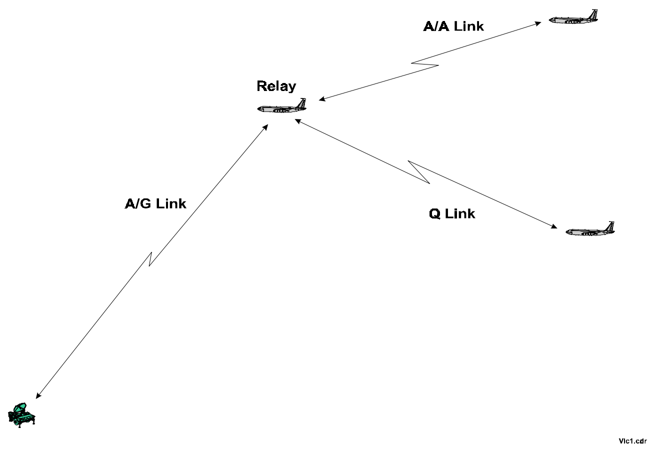

Figure 1 - A Relay Platform

Table 1 shows the primary hardware components at the airframe and link levels. The software packages used to model these hardware components are also shown.

Table 1 High Level Hardware Components

Hardware	Software
Airplane	Platform.ada
High Rate Receive Link	**HR_Rx_Link.ada**
High Rate Transmit Link	HR_Tx_Link.ada
Low Rate Receive Transmit Link	LR_RxTx_Link.ada

Each link has a corresponding set of hardware components used to either transmit or receive data. For instance, a High Rate Receive Link consists of an Antenna, Demodulator, Radio Frequency Assembly, and High Rate Demultiplexer.

Table 2 illustrates the decomposition of the High Rate Receive Link into lower level hardware and software components. For this abstraction, a primitive element, identified by a P, is considered to be fully decomposed from both a hardware and software point of view. Note that the High Rate Receive Module and Low Rate RxTx Module are not primitive elements. Therefore, lower levels of abstraction exist for both of these modules. However, Table 2 does not show the lower levels.

Table 2 High Rate Receive Link Components

Hardware	**Software**	
Airplane	Platform.ada	
High Rate Receive Link	**HR_Rx_Link.ada**	
Radio Frequency Assembly	RFA.ada	P
Antenna	Antenna_Control.ada	P
Demodulator	Mod_Control.ada	P
High Rate Demultiplexer	HR_Demux.ada	
High Rate Receive Module	HR_Receive_MCM.ada	
Low Rate RxTx Module	LR_RxTx_MCM.ada	
Purple Interface Assembly	PIFA.ada	P
Comsec	Comsec.ada	P

4. The Software Design

The hierarchy of the hardware and software, for this system, was illustrated in Tables 1 & 2. Now we'll review the set of operations needed to manage these airborne links. Each Ada package listed in Table 2 provides three primary operations, one to initialize the device, another to gather component status, and a third to command the hardware element. For example, the Radio Frequency Assembly hardware item is implemented by an Ada package called RFA with Initialize, Get Status, and Update operations. These basic operations provide the necessary command and control for a Radio Frequency Assembly.

4.1 Data Design

The design of the primary software database follows a pattern similar to the listed set of operations. Initialization, status, and command databases are each implemented by Ada records. These records of records effectively partition the data consistent with the way the hardware is partitioned. The following excerpts from Ada record definitions illustrate the hierarchical design of these three primary system databases. The record structures do not show the complete decomposition of each database down to primitive levels, but they do depict the essence of the database design.

Initialization data for the Platform contains three parts, one for each communication link, as defined by Type Table.

```
Type Table is
  Record
      HR_Rx     :  HR_Rx_Parameters;
      HR_Tx     :  HR_Tx_Parameters;
      LR_RxTx   :  LR_RxTx_Parameters;
  End Record;
```

The HR_Rx component is also a record as defined by Type HR_Rx_Parameters.

```
Type HR_Rx_Parameters is
 Record
    Antenna          :  Ant_Control.Initial_Parameters;     P
    RFA              :  RFA_Pkg.Initial_Parameters;         P
    Mod_Demod        :  Mod_Control.Initial_Parameters;     P
    HR_Demux_CM      :  HR_Demultiplexer_CM.
                        Initial_Parameters;
 End Record;
```

The first three elements of this record are at the primitive level. The types are defined in their respective packages. The HR_Demux_CM element can be decomposed further and the record is defined by Type Initial_Parameters within the HR_Demultiplexer_CM package.

```
Type Initial_Parameters is
 Record
    HR_Rx_MCM     :  HR_Receive_MCM.Initial_Parameters;
    LR_RxTx_MCM   :  LR_Rx_Tx_MCM.Initial_Parameters;
    PIFA          :  Purple_IF.Initial_Parameters;        P
    Comsec        :  Comsec.Initial_Parameters;           P
 End Record;
```

The system status and command database for a Platform also contains three parts, one for each link. The Platform Status is defined in the following record.

```
Type Table is
Record
      HR_Rx     :  HR_Rx_Status;
      HR_Tx     :  HR_Tx_Status;
      LR_RxTx   :  LR_RxTx_Status;
 End Record;
```

The status component for the High Rate Receive Link is defined by Type HR_Rx_Status**;**

```
Type HR_Rx_Status is
Record
     Antenna          :  Ant_Control.Status;          P
     RFA              :  RFA_Pkg.Status;              P
     Mod_Demod        :  Mod_Control.Status;          P
     HR_Demux_CM      :  HR_Demultiplexer_CM.Status;
 End Record;
```

The first three elements of this record are at the primitive level, but the HR_Demux_CM component is defined by Type Status exported by the HR_Demultiplexer_CM package.

```
Type Status is
Record
    HR_Rx_MCM      :  HR_Receive_MCM.Status;
    LR_RxTx_MCM    :  LR_Rx_Tx_MCM.Status;
    PIFA           :  Purple_IF.Status;        P
    Comsec         :  Comsec.Status;           P
End Record;
```

Type UCF_Table defines the command data for a High Rate Receive Link.

```
Type UCF_Table is
Record
    Antenna       :   Ant_Control.UCF_Elements;   P
    RFA           :   RFA_Pkg.UCF_Elements;       P
    Mod_Demod     :   Mod_Control.UCF_Elements;   P
   HR_Demux     :   HR_Demultiplexer_CM.
                    UCF_Elements;
End Record;
```

The HR_Demux component of the command database is defined by Type UCF_Elements exported by the HR_Demultiplexer_CM package.

```
Type UCF_Elements is
Record
   HR_Rx_MCM     :  HR_Receive_MCM.UCF_Elements;
   LR_RxTx_MCM   :  LR_Rx_Tx_MCM.UCF_Elements;
   PIFA          :  Purple_IF.UCF_Elements;     P
   Comsec        :  Comsec.UCF_Elements;        P
End Record;
```

4.2 Program Call Sequence

Thus far we have examined the basic operations provided by the Ada packages for command and control of the hardware elements and reviewed the structure of the system databases.

Next we'll illustrate the sequence of calls initiated when Platform.Initialize is called from Link_Process which is the main program unit. Note that the applicable component of the system initialization record is passed to the Initialize operation of each package. From Platform the Initialize routine of each link is called and for illustration the hierarchy of calls for the HR_Rx_Link is listed in further detail.

From Link_Process …

```
Platfrom.Initialize
     (The_Platform   =>  Platform_Communication_Element,
      Using_OST      =>  Operational_State_Table.all);
```

The Initialize operation of Platform calls the Initialize operation of each the three links. But, for expediency, only one call is shown at this level.

```
HR_Rx_Link.Initialize
    (The_HR_Rx_Link  =>  The_Platform.HR_Rx_Component,
     With_Parameters   =>  Using_OST.SIT.HR_Rx);
```

In the above routine the Initialize operation for each component of the High Rate Receive Link is called as shown below.

```
Ant_Control.Initialize                    P
    (The_Antenna       =>  The_HR_Rx_Link.Antenna,
     With_Parameters  =>  With_Parameters.Antenna);

RFA_Pkg.Initialize                        P
    (The_RFA           =>  The_HR_Rx_Link.RFA,
     With_Parameters  =>  With_Parameters.RFA);

Mod_Control.Initialize                    P
    (The_Mod_Control   =>  The_HR_Rx_Link.Mod_Demod,
     With_Parameters     =>  With_Parameters.Mod_Demod);
```

```
HR_Demultiplexer_CM.Initialize
   (The_HR_Demultiplexer_CM   =>
                  The_HR_Rx_Link.HR_Demux_CM,
   With_Parameters             =>
                  With_Parameters.HR_Demux_CM);
```

From the Initialize routine of a High Rate Demultiplexer the Initialize operation of its subparts are called.

```
Purple_IF.Initialize                    P
     (The_PIFA          =>  The_HR_Demultiplexer_CM.PIFA,
      With_Parameters  =>  With_Parameters.PIFA);

High_Rate_Receive_MCM.Initialize
    (The_HR_Receive_MCM   =>
              The_HR_Demultiplexer_CM.HR_Rx_MCM,
     With_Parameters    =>  With_Parameters.HR_Rx_MCM);

 LR_Rx_Tx_MCM.Initialize
    (The_LR_Rx_Tx_MCM  =>
                  The_HR_Demultiplexer_CM.
                  LR_RxTx_MCM,
     With_Parameters             =>
                  With_Parameters.LR_RxTx_MCM);

Comsec.Initialize            P
    (The_Comsec        =>
              The_HR_Demultiplexer_CM.Comsec,
     With_Parameters    =>   With_Parameters.Comsec);
```

Similar call structures exist for the Get_Status and Update operations. These hierarchy of calls are initiated by Platform.Operate. When the Operate routine of Platform is called, Operate calls the corresponding Operate routine for the links which in turn call Get_Status and Update respectively. Get_Status at the link level results in the cascade of calls down to the Get_Status operation of each primitive hardware configuration item. In a similar fashion, calling Update at the link level results in a cascade of calls down to the Update operation of each primitive hardware configuration item.

5. Utility for System Modification

Design of this hardware system is modular with specific system functionality mapped into specific components. This building block approach facilitates adding new functionality to an existing system or designing new systems with differing personalities. For example, a High Rate Receive Link becomes a High Rate Transmit Link by replacing the High Rate Receive Common Module with a High Rate Transmit Common Module. That is, replace one Application Specific Integrated Circuit (ASIC) with another. Since the same modular approach was used to design the software for command and control of the system, adding or removing hardware components of the system results in adding or removing the corresponding software components.

As an example, I'll illustrate this modular approach of updating the software to add a Synthesizer component to the High Rate Receive Link. First a Synthesizer package must be written providing Initialize, Get_Status, and Update operations and the components of the Initialize, Status, and Command databases must be defined.

Next the three primary system databases are updated by adding the respective Synthesizer components to these records.

```
Type HR_Rx_Parameters is
Record
    Antenna              :  Ant_Control.Initial_Parameters;
    RFA                  : RFA_Pkg.Initial_Parameters;
    Mod_Demod            :  Mod_Control.Initial_Parameters;
    Synth                :  Synth_Control.Initial_Parameters;
    HR_Demux_CM :  HR_Demultiplexer_CM.
                          Initial_Parameters
End Record;

Type HR_Rx_Status is
Record
     Antenna              :  Ant_Control.Status;
     RFA                  : RFA_Pkg.Status;
     Mod_Demod            :  Mod_Control.Status
     Synth                :  Synth_Control.Status
      HR_Demux_CM  :  HR_Demultiplexer_CM.Status;
End Record;

Type UCF_Table is
Record
     Antenna        :   Ant_Control.UCF_Elements
     RFA            :   RFA_Pkg.UCF_Elements
     Mod_Demod   :   Mod_Control.UCF_Elements
     Synth          :   Synth_Control.UCF_Elements
     HR_Demux     :   HR_Demultiplexer_CM.UCF_Elements;
End Record;
```

Finally, calls to Initialize, Get_Status, and Update for the Synthesizer are added to the corresponding operations in the HR_Rx_Link package. Adding the call to Synthesizer.Initialize within the Initialize operation for a High Rate Receive Link is shown. For the sake of expediency, the call parameters aren't shown. However, as depicted in section 4.2 the appropriate component of the Initialization database is passed by Initialize at the link level to the respective lower level components.

```
Package body HR_Rx_Link is
  Procedure Initialize is
   Begin
        Ant_Control.Initialize
        RFA_Pkg.Initialize
        Synth_Control.Initialize
        Mod_Control.Initialize
        HR_Demultiplexer_CM.Initialize
    End Initialize;  …
End HR_Rx_Link;
```

6. Concluding Remarks

This paper provided a brief discussion of a military communication system and illustrated the mapping between the hardware and software components. Each hardware component was encapsulated by an Ada package, and the package provided Initialize, Get_Status, and Update operations for command and control of the device. Each package also contained type definitions for its component of the Initialization, Status, and Command databases.

Addition or deletion of hardware devices to a communication system requires the addition or deletion of the appropriate software components from the corresponding software system. This modular approach facilitates designing new communication systems or modifying existing systems.