7. ANKARA INTERNATIONAL AEROSPACE CONFERENCE
11-13 September 2013 - METU, Ankara TURKEY

AIAC-2013-019

API FOR DDS BASED MIL-STD-1553B BUS SIMULATOR

Sema Çam1
Aydın Yazılım ve
Elektronik AŞ.
Ankara, Turkey

Umut Durak2
Roketsan Missiles Inc.
Ankara, Turkey

Ertan Deniz3
ESEN System
Integration
Ankara, Turkey

Halit Oğuztüzün4
Middle East
Technical University
Ankara, Turkey

ABSTRACT

MIL-STD-1553B Serial Data Bus standard which defines a Digital Time Division Command/Response Multiplexed Data Bus to connect avionic system components on an aircraft. The standard requires specialized and expensive physical layer hardware for the bus protocol to operate on. Thus, a cost-effective simulator with common networking infrastructure and hardware is desirable. Application Programming Interface (API) for MIL-STD-1553 Bus Simulator, proposed in this paper, provides a simplified interface to implement MIL-STD-1553 Serial Data Bus without its mechanical and electrical characteristics. The bus simulator has been developed based on the Data Distribution Service (DDS) middleware standard. The API on the other hand implements word structure and format of the 1553B standard with its two functional modes of terminals by using Data Centric Publish Subscribe (DDS/DCPS) layer of DDS. Only three functional modes of terminals allowed on the data bus: the bus controller, the bus monitor, and the remote terminal. However, bus monitors do not take part in data transfers; thus, they are not in the scope of this study. API aims to isolate the networking interface from the user interface without interfering with network programming. The first (and the top) layer of the architecture is for interfaces to initialize Bus Controller, Remote Terminals and their messages. The second layer gets the messages from terminal interfaces and enables transmitting and receiving messages on DDS/DCPS. Finally, the third layer consists of DDS generated applications for Publish/Subscribe network.

INTRODUCTION

Application Programming Interface for Data Distribution Services based MIL-STD-1553B Bus Simulator provides an interface to implement 1553B Serial Data Bus Simulator. The application references 1553B standard without its hardware and wiring components [1]. While timing requirements of the word transmissions are not implemented, word structure and format of the standard's bus protocol and two functional modes of bus controller and the remote terminals are simulated. Thus, the API helps developers who are dependent on 1553B standard in communication and need to test their messages, including Command, Status and Data Words of the 1553B Data Bus without the standard's time restriction. This should be helpful in the pre-SIL phase of system integration and testing. The background for the related portions of the standard and the key implementation techniques of the API are provided in the following sections.

[1]Researcher, Email: semac@ayesas.com
[2]Unit Head, Email:udurak@roketsan.comtr.tr
[3]Principal Software Engineer, Email:ertan.deniz@esensi.com.tr
[4]Assoc.Prof., Email: oguztuzn@ceng.metu.edu.tr

BACKGROUND

MIL-STD-1553B Standard

MIL-STD-1553B is a widely used avionic bus specification which defines a bus protocol. It includes specifications for terminal device operation and coupling, word structure and format, messaging protocol and electrical characteristics for a data bus.

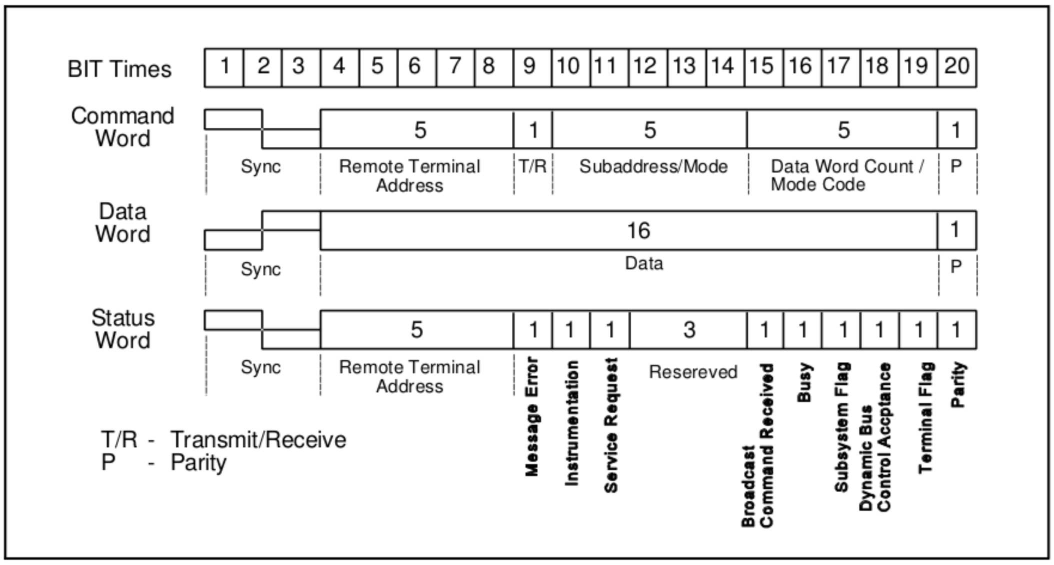

Figure 1: *MIL-STD-1553B Word Formats [4]*

MIL-STD-1553B Bus Protocol: MIL-STD-1553B Bus Protocol includes three functional modes of terminals: Bus Controller (BC), Remote Terminals (RT) and Bus Monitors (BM). A BC controls all the RTs as master control and there can be up to 31 (thirty-one) RTs connected to a single bus. There may also be one or more passive Bus monitors in the bus topology as they do not transmit any messages [2]. A typical example can be as follows. A fighter platform may have a central fire control computer which acts as BC and fire control radar and the inertial measurement unit joins the bus as RTs.

Bus controller transmits command words and receives status words from remote terminals. Remote terminals send status words and receive command words from the bus controller. Both send and receive data words. The standard's word formats can be summarized as depicted in Figure 1.

The first three bit times of all the word types are sync field. According to Condor Engineering Inc. (2000), "the use of this distinct pattern allows the decoder to re-sync at the beginning of each word received and maintains the overall stability of the transmissions" (p. 21) [3]. Command Word keeps five bits of target Remote Terminal Address (bit times 4-8) and 1 bit (T/R) to declare to the target RT to transmit or receive information. The next 5 bits are for Subaddress or Mode Commands (bit times 10-14). Logic "00000" or "11111" indicates that the command is a Mode Code Command. All other logic combinations of this field are used to transmit/receive the data to/from different functions within the subsystem. The next five bits (bit times 15-19) define the Word Count or Mode Code to be transmitted or received [3]. If the Subaddress/Mode Code field is Mode Code, then the next five bits indicate the Mode Code to be performed. If it is not a mode code, the field indicates the number of data words to be transmitted or received according to T/R bit value. As it is 5 bits long, there can be 32 data words. As for the last bit of all words, it is word parity bit and only odd parity is used.

The Data Word has 16 bit information which can be specified by the designer. The only requirement is the most significant bit (MSB) of the data word [3].

Status Word has 5 bits terminal address which must be matched with Command Word 5 bits Remote Terminal Address. Message Error bit aims to inform about error detection or invalid message to the terminal. The Instrumentation bit (bit time 10) is provided to differentiate between a command word and a status word as they have the same sync pattern. The Service Request bit (bit time 11) is defined so that the remote terminal can inform the bus controller that it requires service [3]. Reserved bits (bit times 12-14) are reserved for future in case of the change of the standard. Broadcast Command Received bit (bit time 15) is set when a valid broadcast message is received. As for busy bit (bit time 16), it is used when the remote terminal is unable to exchange data between the remote terminal and the subsystem even though a command from the bus controller is received [3]. Subsystem flag (bit time 17) confirms the subsystem that the remote terminal is connected. Dynamic Bus Control Acceptance Bit (bit time 18) informs the bus controller about receiving the Dynamic Bus Control Mode Code and accepting the control of the bus. Terminal Flag (bit time 19) informs the bus controller when a fault or failure within the remote terminal circuitry occurs [3].

Furthermore, there are three basic types of information transfers that are defined by MIL-STD-1553B: Bus Controller to Remote Terminal transfers, Remote Terminal to Bus Controller transfers and Remote Terminal to Remote Terminal transfers. The other transfers are listed below:

- Mode command without data word transfers
- Mode command with data word (transmit) transfers
- Mode command with data word (receive) transfers
- Bus controller to remote terminal(s) (broadcast)
- Remote terminal to remote terminal(s) (broadcast)
- Mode command without data word (broadcast)
- Mode command with data word (broadcast) [5]

METHOD

Development Environment

DDS standard [6] defines a middleware with a data centric and publish-subscribe architecture for real-time system [7]. It is capable of using shared memory Publish Subscribe architecture where data is available in subscriber's data cache that can be content filtered. OpenSplice DDS [8] which is a well-accepted implementation of the Object Management Group's (OMG) DDS for Real-Time Systems messaging middleware standard [8] is preferred as the DDS implementation as the tool is open source and has a good documentation. Moreover, the API for DDS Based MIL-STD-1553B Bus Simulator is developed by using Java programming language.

The middleware requires the specification of the data interfaces in Interface Definition Language (IDL). An IDL is a language that is used to define the interface between a client and server process in a distributed object system. Each interface definition language also has a set of associated IDL compilers, one per supported target language. An IDL compiler compiles the interface specifications, listed in an IDL input file, into source code that implements the low level communication details required to support the defined interfaces [9]. IDL file of OMG DDS is compiled in order to generate code for DDS/DCPS specialized interfaces (TypeSupport, DataReader and DataWriter) [10]. Therefore, the IDL file is developed along with the API.

Design

Object oriented design patterns are utilized in API design. First, Singleton design pattern is adopted in order to constrain the number of Bus Controller instances to one. Second, Factory design pattern is used for bus controller and remote terminal to distinguish which word to create. The pattern allows Bus Controller to create only command words and remote terminals only to create status words. The relations between the classes are pictured in the UML class diagram that is provided in Figure 2. All the other classes that are generated by compiling DDS IDL files for topic types such as DataReader, DataWriter and DataListener and these classes' methods are not illustrated in the diagram. However, they are instantiated in the classes below and mentioned in the summary of the diagram. IBusController, BCApp, IRemoteTerminal, RTApp classes and Command Word, Status and Data Word classes serves for the first layer of the API architecture as the interface. Other classes in the diagram aim to separate DDS and user interface.

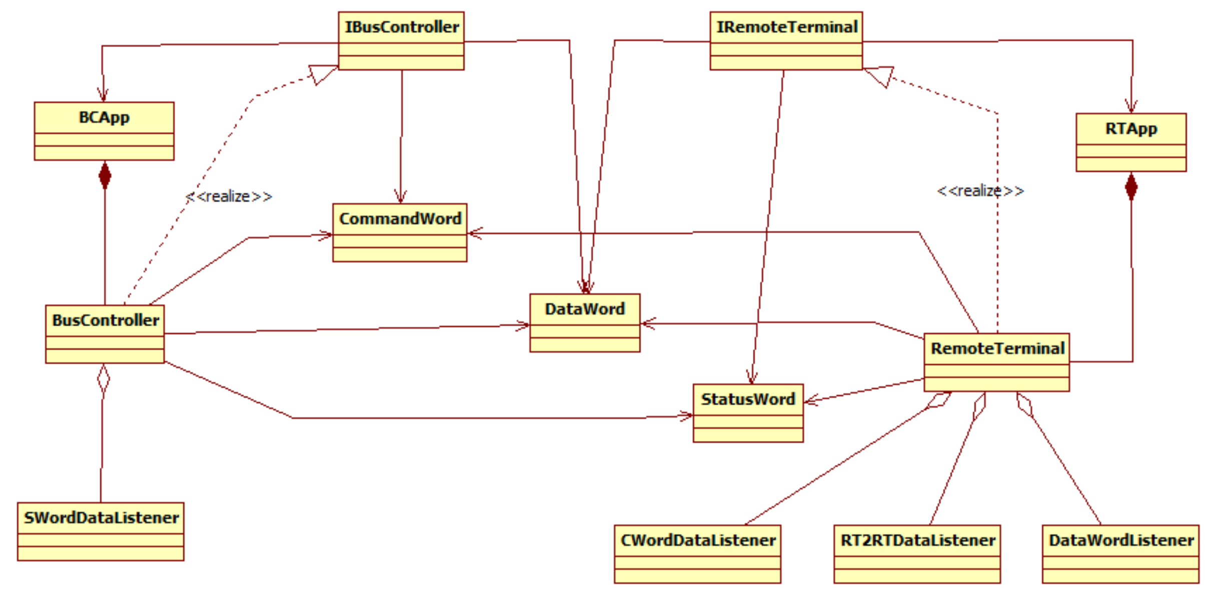

Figure 2: *Top Level Class Diagram of API*

The classes in the diagram are summarized below. In the diagram uni-directional association which is shown with an open arrowhead is used for two classes that are related and there is some dependency between like BusController and CommandWord classes. BusController class needs CommandWord class in some methods. On the other hand, CommandWord class is not aware of BusController class. An association with an aggregation relationship indicates that one class is a part of another class. It is indicated as diamond figure. For instance, RemoteTerminal class makes CWordDataListener class wait for Command Word instead of implementing necessary methods itself. Another relation is realization which is shown as a broken line with unfilled arrowhead. BusController and IBusController classes have realization relationship. IBusController is just the implementation of the functionality defined in BusController class. Finally, the composition relationship is just another form of the aggregation relationship which is shown with a filled diamond. When BusController class which has composition relationship with BCApp is destroyed, BCApp class also is obliterated.

IBusController

This class serves as an interface to Bus Controller to reach its methods, and instantiates Command and Data Words that are to be sent by the Bus Controller. Furthermore, Factory Pattern is implemented to allow only Command Word and Data Word creation.

BCApp

The class notifies IBusController class when a Status Word or Status Word with Data Word is read.

BusController

BusController handles all the transfers between DDS and IBusController. It gets Command and Data Words from IBusController. Then, the class creates DDS Topic for Command Word that its type is already defined by IDL file and Data Writer to write Command Word as publisher. However, Command Word of Remote terminal to remote terminal (RT2RT) and Data Words that follow a Command Word have different IDL file for topic type. Because, not all the Command Word type can be written by creating the same topic as this type of word needs additional information of remote terminal that the message to be sent. As for Command Word with Data Word, it is written adjacent to its Command Word instead of writing the Data Word and a Remote Terminal address as a topic type after writing Command Word. Data and Command Word separation is not implemented to avoid creating traffic and writing terminal address twice. Therefore, another topic and Data Writer are created for these types of Command Word. It also reads Status Words by the help of SWordDataListener class that implements DDS Data Listener as subscriber. Whenever a Status Word is read, SWordDataListener class notifies BusController class. Besides, the class is applied Singleton Pattern to restrict its more than one instantiation.

SWordDataListener
It is a listener class for BusController class that waits for Status Words from Remote Terminals and whenever a Status Word is read, it notifies BusController class.

IRemoteTerminal
This class serves as an interface to Remote Terminal with an id to reach its methods and instantiates Status and Data Words that to be sent by the Remote Terminal. Furthermore, Factory Pattern is implemented to let only Status Word and Data Word creation.

RTApp
The class notifies IRemoteTerminal class when a Command Word or Command Word with Data Word is read.

RemoteTerminal
This class handles all the transfers between DDS and IRemoteTerminal like BusController. It gets Status and Data Words from IRemoteTerminal. Then, the class creates DDS Topic for Status Word that its type is already defined by IDL file and Data Writer to write Status Word as publisher. Additionally, topic type for RT2RT communications that includes Data Word and a Remote Terminal address data is implemented as publisher. RemoteTerminal class subscribes to topics for Command Word, Command Word with Data Word and RT2RT communication words. The class both reads and writes data for RT2RT communications.

CWordDataListener
It is a listener class for RemoteTerminal class that waits for Command Words from Bus Controller and whenever a Status Word is read, it notifies RemoteTerminal class.

RT2RTDataListener
It is a listener class for RemoteTerminal class that waits for Command Words for RT2RT communications from Bus Controller and whenever a RT2RT Word is read, it notifies RemoteTerminal class.

DataWordListener
It is a listener class for RemoteTerminal class that waits for Data Word from another Remote Terminal and whenever it is read, it notifies RemoteTerminal class.

CommandWord
This class is the definition of a Command Word.

StatusWord
This class is the definition of a Status Word.

DataWord
This class is the definition of a Data Word.

Other mentioned critical implementation decision is using OpenSplice DDS as networking middleware. Real-time distributed systems expect predictable distribution of data with minimal overhead. Since the resources are limited and each data may have different resource requirements, it is essential for a middleware to have configuration parameters [7]. DDS defines these configuration parameters as QoS (Quality of Service). The wide range of QoS parameters allows DDS middleware to be configured according to the specific requirements of each component in the system. DDS middleware is fully aware of the data content that is being transmitted between applications. This awareness is provided by DDS Topic. Applications publish data with a DDS Topic and only those applications that subscribed to that Topic Type read this data. It is a characteristic feature of data centric architecture that distinguishes it from message centric or service based architectures. Applications specify only the data they are willing to receive and the middleware filters unnecessary data. This content based filtering is achieved by a subset of SQL [10]. A DataReader associated with a ContentFilteredTopic receives only the desired data specified in the ContentFilteredTopic creation. Namely, ContentFilteredTopic also constricts the data to be read for that Topic with a SQL. On the other hand, a DataReader associated with a normal Topic will receive all data published to that Topic, but applications can later query the DataReader cache with SQL [7]. Therefore, command words are associated with a ContentFilteredTopic in order to be received by addressed remote terminal.

Architecture

The API for DDS Based MIL-STD-1553B Bus Simulator is built upon three layered architecture. The first layer includes the distributed interfaces of the Bus Controller and Remote Terminals. Both of the interfaces publish words and send them to concerned terminal objects in the second layer. The second layer objects transmit words to third layer where publish-subscribe communications occur. In the third layer, network communications according to 1553B Bus Protocol are implemented by using DDS features. DDS/DCPS is used in this layer by addressing terminals. The three layered architecture is shown in Figure 3.

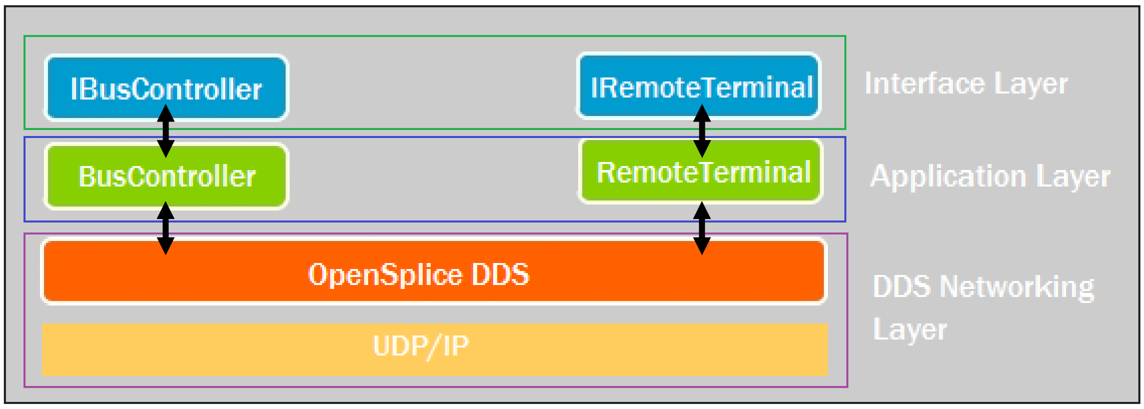

Figure 3: *Three layered architecture of the API*

The architecture provides the separation between the application interface and the message transmitting network layer by employing classes such as BusController and RemoteTerminal for necessary DDS method invocation. This separation is provided by the application layer which is in the middle. The interface layer for both Bus Controller and Remote Terminals is only employed for data insertion. Interface of the Bus Controller receives the Command Words to be sent to remote terminals. Data words are also received with Command Words. On the other hand, Interfaces of Remote Terminals allow initiating Status Words with or without Data Words. When the words are set in the interface of the terminals, they are sent to application tier of the architecture. The users of the API do not need to deal with the rest of the communication. The application tier handles the networking and data handling.

After the application tier gets the words, Bus Controller begins to transmit the Command Words by using DDS and waits for Status Words from the corresponding Remote Terminals. Data Writer and Data Readers for both Command and Status Words participate in the bus simulation to transmit and receive data by DDS. Concurrently, every instantiated Remote Terminal subscribes to Command Word that is addressed to itself from the Bus Controller. If a Command Word is received by the Data Reader of a Remote Terminal, according to Command Word type, the terminal writes Status Word or Data Word to be received by Bus Controller or another Remote Terminal. Recall that only the Bus Controller receives the Status Words. In Remote Terminal to Remote Terminal transfers, a remote terminal waits for Data Word from another terminal.

All the network communications occurs in the third layer of the API architecture. DDS provides the communication of distributed applications such as Bus Controller and Remote Terminals. The middleware helps to create Data Listeners, Data Readers and Data Writers of each distributed applications with a command. The middle tier of API uses this listeners, writers and readers in order to transmit published data or receive subscribed data. Additionally, the content filtering enables transmission of the necessary data to only addressed terminal. The abstraction of networking is accomplished by separating the networking and user interface with an application tier that involves Bus Controller and Remote Terminal class implementations.

DISCUSSION AND CONCLUSION

This work provides a programming interface for MIL-STD-1553B Bus Simulator programmers. The interface enables a straightforward and commonly available environment to simulate the 1553B Standard's Bus Protocol.

The API requires scenario based inputs for word transmission testing. Necessary input can be provided by defining XML based interface scenarios. A separately developed test tool that supports MIL-STD-1760 based test scenarios represents an example usage of the API. The tool provides user interfaces that can be converted to XML file. The interface enables creation of Command, Status Word and Data Word of MIL-STD-1553B standard and Data Words are defined according to MIL-STD-1760 standard. Additionally, the tool, enables easy generation of store control, store monitor, store description and aircraft description standard messages, well contributes to the development of test scenarios regarding safety critical control and monitor functions of conventional weapons [11].

Most importantly, the API eliminates the need for MIL-STD-1553B special and expensive hardware and wiring components in the early phases of system integration and testing.

Additionally, distributed data communications of Bus Controller and the Remote Terminals are implemented with a network abstraction layer by using the publish/subscribe middleware DDS. DDS seems to be a good choice for the middleware mainly because of its content filtering features.

Time requirements of MIL-STD-1553B standard for message transmitting and receiving can be an aim for a complete simulation in a real time operating system environment.

References

[1] Downing, N., October 2006, "Virtual MIL-STD-1553," 25th Digital Avionics Systems Conference, 2006IEEE/AIAA, p: 1.

[2] Alta Data Technologies, 2007, *MIL-STD-1553 Tutorial and Reference*.

[3] Condor Engineering, Inc., June 2000, MIL-STD-1553 Tutorial, *Word Types*, p:19-27.

[4] US Department Of Defense, 1978, *MIL-STD-1553B Aircraft Internal Time Division Command/Response Multiplex Data Bus*.

[5] ILC Data Device Corporation, 2003, *MIL-STD-1553 Designer's Guide*, 6th ed..

[6] Object Management Group (OMG), 25 July, 2013, *Data Distribution Service (DDS)*, http://www.omg.org/hot-topics/dds.htm, Retrieved on 26 July, 2013.

[7] Object Management Group (OMG), 2007, *Data Distribution Services for Real-time Systems*

Version 1.2.

[8] PrismTech, 2013, *OpenSplice DDS*, http://www.prismtech.com/opensplice, Retrieved on 21 July, 2013.

[9] A. David McKinnon, Interface Definition Language, http://csis.pace.edu/~marchese/CS865/Papers/interface-definition-language.pdf, Retrieved on 21 July, 2013.

[10] Object Management Group (OMG), 2011, *Interface Definition Language (IDL) Specification*,

Version 3.5.

[11] Güçlü, K., *Testing a MIL-STD-1553 Bus Simulator with MIL-STD-1760 Based Test Scenarios*, SE 599 Term Project Final Report, Middle East Technical University, May 25, 2012.