

# **Designing an Object-Oriented Data Processing Network**

**Hsueh-szu Yang, Software Engineer**  
**Nathan Sadia, Director of Ground Software**  
**Benjamin Kupferschmidt, Technical Manager**  
Teletronics Technology Corporation

## **Abstract**

There are many challenging aspects to processing data from a modern high-performance data acquisition system. The sheer diversity of data formats and protocols makes it very difficult to create a data processing application that can properly decode and display all types of data. Many different tools need to be harnessed to process and display all types of data. Each type of data needs to be displayed on the correct type of display. In particular, it is very hard to synchronize the display of different types of data. This tends to be an error prone, complex and very time-consuming process.

This paper discusses a solution to the problem of decoding and displaying many different types of data in the same system. This solution is based on the concept of a linked network of data processing nodes. Each node performs a particular task in the data decoding and/or analysis process. By chaining these nodes together in the proper sequence, we can define a complex decoder from a set of simple building blocks. This greatly increases the flexibility of the data visualization system while allowing for extensive code reuse.

## **Keywords**

Ground Station Software, Data Processing, Data Analysis, Data Reporting and Exporting, Object Oriented

## **Introduction**

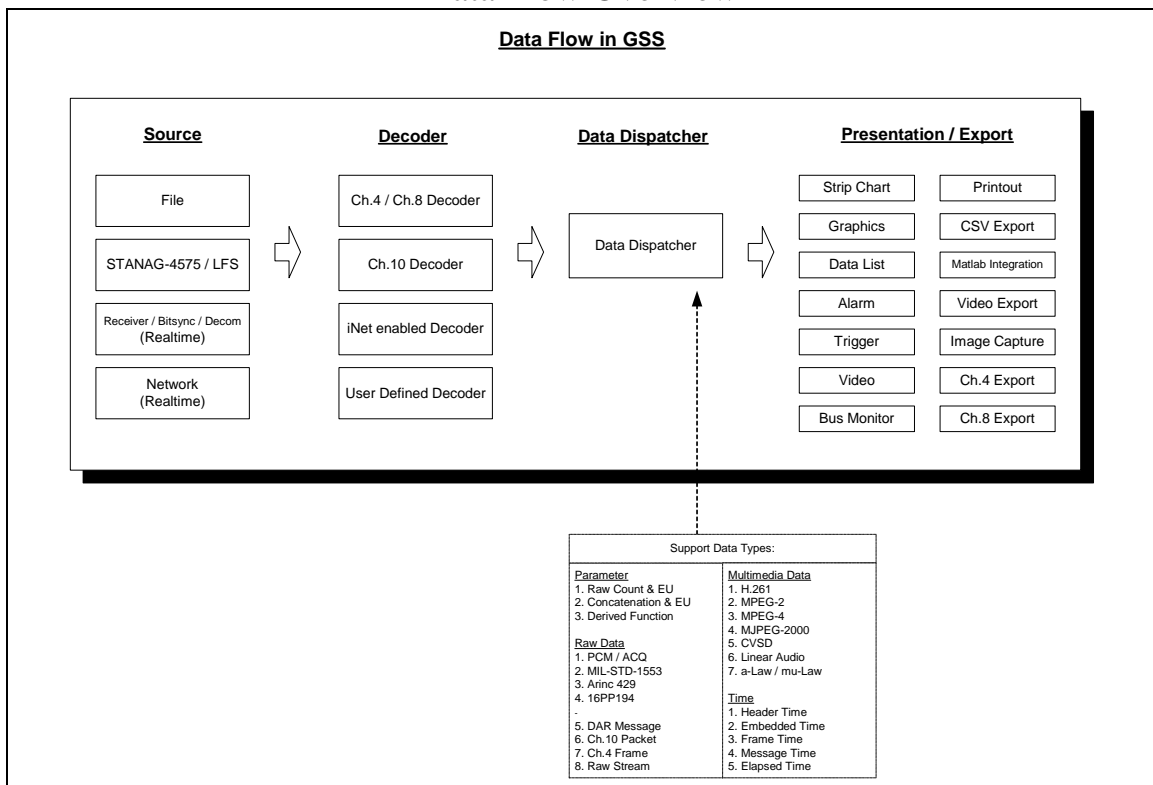
Advancements in communication technology have led to the development of many different protocols and encoding systems for various applications. In order to decode and extract information from each protocol, the data processing system needs to have a set of data processors for each format. While the system that encodes the data only needs to implement a small number of protocols, the data processing system is required to deal with many different protocols and data formats. An example of a data processing system

is the receiver side of a ground station. The sheer variety of protocols and data formats is one of the largest challenges involved in creating a data processing system.

In this paper, we propose the use of object-oriented programming (OOP), which is a widely used software development concept, to organize the software's code and reduce the complexity of dealing with multiple data formats. Instead of writing many different decoders to process each data format from beginning to the end, we divide the decoding process into several smaller tasks. Each task is encapsulated into a class. This is the essence of the OOP strategy for dividing large problems into several smaller problems.

The inheritance feature in OOP will be very useful for organizing the tasks. The data formats used in communication technologies are usually layered and the differences between data formats are often quite small. Inheritance enables programmers to reuse source code by allowing them to create a base object that contains a common set of code that is used by all of the data formats. This base object includes functions, which can be overridden by each object that is derived from the base. This helps to organize and simplify the task of implementing a large system. It also makes the system easier to maintain.

## Data Flow Overview



**Figure 1: Data Flow In TTC's GSS Software**

Figure 1 depicts an overview of data flow on the receiver side of TTC's ground station system (GSS). The system is roughly divided into four layers that are executed from the left to the right. The four layers are the Source, the Decoder, the Data Dispatcher and the

Presentation / Export layer. In each layer, there is a collection of components that can process different types of inputs. After processing their inputs, these components pass the output data stream to the next layer. Typically the data flows from the left to the right. Some of the components have the ability to extract a data stream. This stream can be rerouted back to an earlier stage in the system so that it can be processed. In effect, the system can create branches to process multiple streams.

The first layer is called Source. This layer contains components that retrieve data from physical devices. Supported devices include all of the following:

- A standard data file on a PC's hard drive
- A file on a data Recorder
- A live data stream from a Receiver / BitSync / Decommutator
- Live data from an Ethernet network

Parameter	Raw Count & EU
	Concatenation & EU
	Derived Function
Raw Data	PCM / ACQ
	Mil-Std-1553
	Arinc-429
	16PP194
	DAR Message
	Chapter 10 Packet
	Chapter 4 Frame ( Pack / Unpack / Throughput )
	MAC / UDP / TCP / IP
	Raw Byte Stream
	User Define
Multimedia Data	H.261
	MPEG-2
	MPEG-4
	MJPEG-2000
	High Definition MJPEG-2000
	CVSD
	Linear Audio
	a-Law / mu-Law
Time	Header Time
	Embedded Time
	Frame Time
	Message Time
	Elapsed Time

**Figure 2: Supported Data Types In GSS**

The second layer decodes and processes data. There are sub-layers inside that handle the details of interpreting each data format. A more detailed description of these sub-layers

follows in a later section. One of the topics that will be discussed is how we apply object-oriented programming to make data processing more flexible and organized.

The Data Dispatcher layer dispatches data from the output of the decoder layer. One of biggest advantages of using a dispatcher is that it is loosely coupled with the decoder. This means that the data consumers in the final Presentation and Export layer are decoupled from the decoder. This has many benefits including faster development, easier debugging of problems and the presentation of a standardized interface for each data consumer to use to get data. This makes it much easier for teams to develop components simultaneously.

The table in Figure 2 lists the data types that are supported by TTC's GSS software. The list can be easily extended thanks to the power and flexibility of the multi-layered software design.

## Object-Oriented Programming (OOP)

Object-Oriented Programming is based on four main principles:

### **Abstraction**

Abstraction is a process for simplifying complex problems by creating objects that model the various layers of the problem. This allows each aspect of a problem to be addressed at the appropriate level in the model.

### **Encapsulation**

Encapsulation conceals the functional details of an object. This allows objects to be treated as black boxes. Their internal code and functions can change as long as their external interface remains the same.

### **Inheritance**

Inheritance is a concept that allows users to create derived objects that inherit the functionality of their parents. These derived objects can be extended to include additional features that are not present in their parent object.

### **Polymorphism**

Polymorphism allows programmers to interact with derived objects as if they were instances of their parent object.

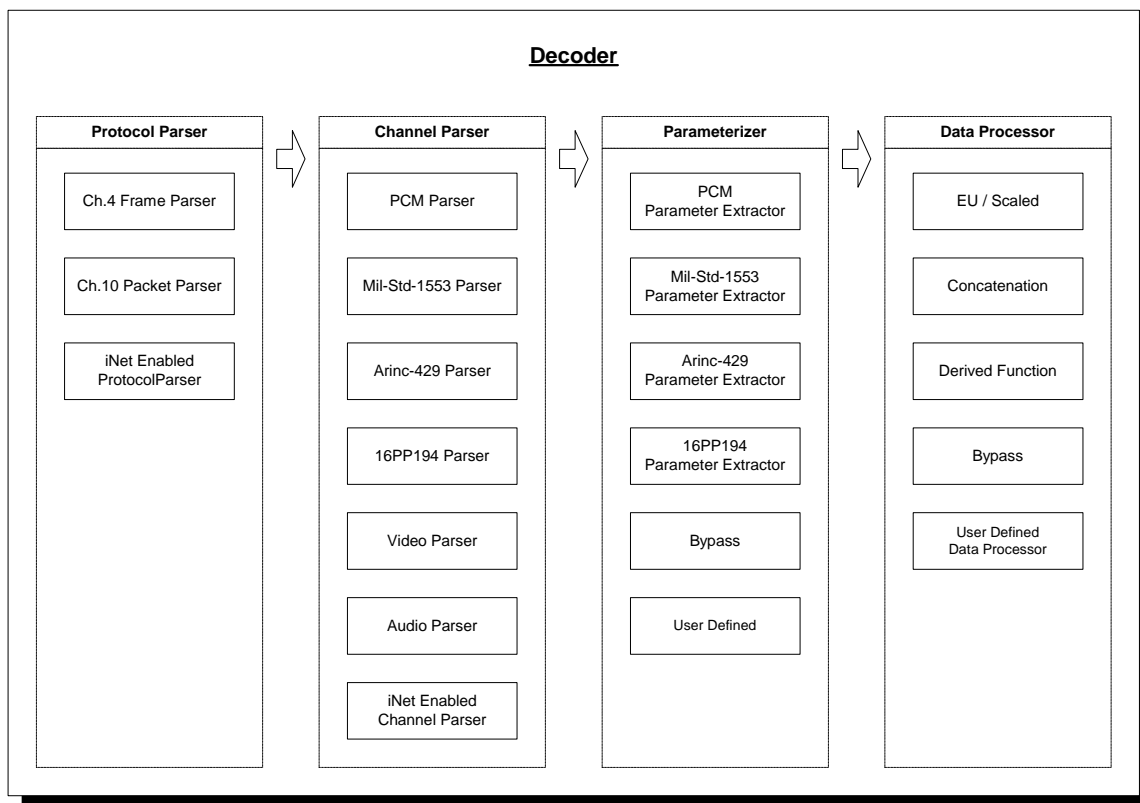
Here is a simple example of Object-Oriented Programming. Let's suppose that a program contains a base object called Food. Food exposes a function called Eat. A programmer can create inherited objects that are based on the Food object. For example, the programmer could create two objects called Apple and Orange. These objects inherit from Food so they also contain the Eat function. Thanks to OOP, another part of the program that doesn't know anything about Apple and Orange can still use them as long as it understands that the Eat function can be used on any Food object. The exact implementation details of Orange and Apple don't matter because they are being used as

black boxes. This model of Food is a simple example of how abstraction allows the user to simplify a big problem into a set of easy-to-use objects.

## Object-Oriented Data Processing

This paper describes a way to process data that uses and extends the Object Oriented Programming concept. This makes it possible to:

- Divide the dataflow pipeline into multiple small tasks. Each task is encapsulated as an object.
- Abstraction is used to create classes that contain the decoder for each real-world data format.
- Each layer in the system starts with a base class that implements standard functionality. The variations and unique features of each data format are defined in the inherited classes.
- Polymorphism makes it possible for each layer to interface with the objects from the other layers without having to know how they work. As long as each layer knows about the functions that are available on the base class of the other layers then the system can form a data pipeline without having to know the implementation details of each class.



**Figure 3: The Data Decoder In TTC's GSS Software**

The data decoder is the place where data formats are interpreted and useful information is extracted. We propose two possible models for the data decoder: the layered model and the network model.

## Data Decoding Using The Layered Model

Figure 3 contains an example of the layered model. In this model, data flows through a series of layers from the left to the right. To process the data stream, one or more parse instances will be created. A parser reads from an input data stream, processes the data, extracts information and converts the result into an output data stream. The data format in the input stream may not be the same as the one in the output stream.

The first layer is the protocol parser, which unpacks data from a byte stream or a bit stream. There are three parsers in Figure 3's example: the Chapter 4 parser, the Chapter 10 parser and the iNET-enabled parser. One of the major tasks for the protocol parser is to locate packet boundaries. Synchronization words are a common way to locate the boundary of a packet. For example, Chapter 10 uses 0xEB25 as its sync word, while Chapter 4 allows users-defined sync words.

In order to read data from a bit stream, it is necessary to perform byte alignment. A bit stream is just like a serial string of bits, 0's and 1's. It is a challenge to identify which bit is the first bit in a word, and still keep the system running in real-time.

Both Chapter 4 and Chapter 10 are sub-classes of the protocol parser so they perform similar functions while deviating in a number of ways. The Chapter 4 frame may reside in a byte stream or in a bit stream. Thus, we need two Chapter 4 frame parsers. One processes the classic PCM frame and another one processes Throughput Chapter 4 frames. The Throughput Chapter 4 parser inherits from the standard Chapter 4 frame parser. The major task of the Chapter 4 frame parser is to locate the minor frame boundary by using the sync words. The Throughput parser inherits from the Chapter 4 frame parser and performs one extra task. It aligns the data that it reads from the bit stream on byte boundaries.

There are several other tasks that the protocol parsers perform. They retrieve the time stamp or relative time counter from the data and they verify the data packet's integrity.

The second layer is channel parser. Multiple channel parsers may be created because multiple data streams can be extracted from the protocol parser. Even for Chapter 4 data, it is possible to extract embedded PCM data streams.

PCM is a time-driven protocol and the PCM frame can be represented by a two-dimension array of data words. The major task of the PCM parser is to construct a major frame and verify the Sub-Frame ID word. This prevents bad data from flowing into the next layer of the system. MIL-STD-1553, ARINC-429, ARINC-629, and 16PP194 are message-based protocols. Their parsers must extract message data and accurate message time stamps from the raw byte stream. Extracting video and audio data is

comparatively simple in this layer but there are cases where reprocessing is needed. An example of a type of reprocessing that is often needed for video is byte swapping.

One issue that needs to be considered is that the layout of data for a particular data type may be different depending upon the protocol that the data is embedded in. A typical example is MIL-STD-1553 messages. When the messages are placed in a PCM Format, they are serialized as a byte stream. This is typically called a Chapter 8 PCM Stream. Messages may be mixed together in this format and a 1553 parser for PCM data needs to extract and restore the messages. The 1553 parser for Chapter 10 files is much simpler because the messages are recorded sequentially in the file. By using abstraction, the MIL-STD-1553 parser defines the basic tasks that are required to extract 1553 data. Sub-classes can inherit from the base class for extracting 1553 data from PCM and Chapter 10. This allows the system to easily handle the different layouts.

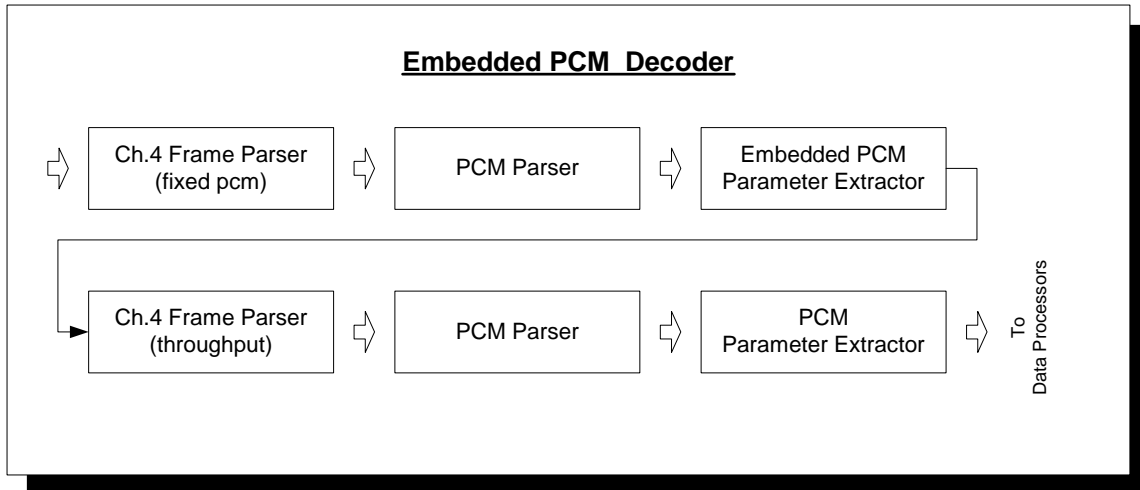
The parameter layer is used to extract meaningful measurements from the raw data format. PCM parameters are defined in a PCM frame based on their word and frame positions. MIL-STD-1553 parameters are defined by their Bus ID, RT Address, Sub-Address, Transmit / Receive and their word position in the message.

After the parameter layer, the data is in a standardized format and is ready for further processing in the Data Processing layer. This layer contains a powerful calculation engine. Engineering Unit conversions, concatenations and derived functions are defined in a mathematical expression. The calculation engine plugs the parameter values into the expression to calculate the results.

The main limitation of the layered model is that the data flows from the left to the right but not the opposite direction. In some cases, a data acquisition system contains a set of serially connected units. This makes it possible to embed one data format inside of another format. Asynchronous PCM streams and video inside a PCM frame are two common examples. The Chapter 8 data stream could also technically be treated as an embedded MIL-STD-1553 or ARINC-429 stream inside of a Chapter 4 PCM stream.

## Data Decoding Using the Network Model

A network model is an alternative that provides a more flexible way of connecting the data path. In a network model, virtually any parser can connect to the output of any other parser. An adaptor may be required if the data types are not compatible between the parsers. For example, consider an asynchronous PCM stream, which is a PCM stream that is embedded within another PCM stream. To process an embedded stream, the parameter data can be redirected from the Parameter parser back into the Chapter 4 frame parser in the protocol layer. This allows the embedded stream to be treated as a distinct data source. An example of this is shown in Figure 4.

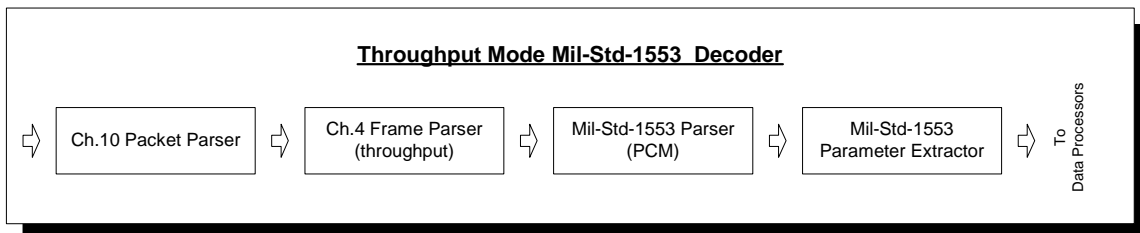


**Figure 4: An Embedded PCM Decoder for a data stream from the DCM-101 card or the MSDI-102-1 module.**

The first component in the embedded PCM decoder is a classical Chapter 4 frame parser. Its task is to locate the frame boundaries and check the packet integrity. The second component is the PCM Parser, which constructs the major frame. The Embedded PCM Parameter Extractor extracts the required data from specific locations in the PCM format, and then forwards the raw data to the Throughput Chapter 4 Frame Parser.

The Throughput Chapter 4 Frame Parser accepts a bit stream from the parameter extractor component. Its task is to align the data on the byte boundaries, locate the frame boundaries and check the data integrity. After the embedded data is extracted, another PCM parser can be used to construct the embedded major frames. The final step is to extract regular PCM parameter from the embedded major frames.

Decoding an embedded PCM stream is very complex. However, thanks to the object-oriented data processor, the decoding scheme becomes much simpler and better organized. As an added benefit only two new components are needed for the decoding pipeline, the other four components are reused from the classical PCM decoder. The two new components are the “Embedded PCM Parameter Extractor” and the “Throughput Chapter 4 Frame Parser.”



**Figure 5: A MIL-STD-1553 Throughput Mode Decoder for a data stream from the XBIM-553-1 card**

Figure 5 shows another example of the ability of the object-oriented data processing system to extract embedded data. The data format in this example is a Chapter 8 channel that is located inside of a Chapter 10 file. The Chapter 8 data is recorded in throughput mode. There are several challenges involved in decoding this data. The first challenge is finding the correct bit alignment in the bit stream. The second challenge is to separate and extract the MIL-STD-1553 messages from PCM stream. This is a challenge because multiple messages may be mixed together in the stream.

The first component is the Chapter 10 Packet Parser, which locates the packet boundaries, checks the packet integrity and calculates the absolute time stamp. The time stamp can be calculated from the relative time counter and a time packet in the Chapter 10 file. The next stage is the Chapter 4 Frame Parser. As discussed in the previous example, this component's task is to align the byte boundaries and locate the minor frame boundaries. The 1553 Parser extracts and separates messages from the PCM stream, while the 1553 Parameter Extractor extracts the raw data words from the messages. All of the components in this example are reused elsewhere in the system. There are no new parsers introduced in this decoder.

## Conclusion

Object-Oriented Data Processing has many benefits. It divides the complex data processing task into several smaller and simpler tasks. This is a classic divide-and-conquer strategy and it is very useful for resolving complex problems. The encapsulation of tasks creates components that can be easily re-used in different data processing scenarios. Inheritance increases the usefulness of each parser, and reduces the maintenance costs at the same time. Abstraction helps to create a simple and clean model, and polymorphism provides the flexibility for different data processing methods.

One consequence of using Object-Oriented Data Processing is that it introduces additional overhead into the system. Fortunately, modern computers are much more powerful and the use of OOP data processing only requires a small amount of CPU time. Furthermore, this solution is ideally suited for a distributed computing environment. If in the future, complex calculations are required, then the solution can be implemented on a distributed computation network. This will allow multiple computers to work together on the calculation.

In recent years, the telecommunication industry is moving faster and faster. As the Internet grows more popular, it also pushes network technology to improve. In order to adapt to these rapid changes and the complexity of the protocols that are used in modern systems, an Object-Oriented Data Processing system is required. This system is an excellent solution to the challenges presented by the need to decode and play data from modern data acquisition systems.

## References

[1] Armstrong, The Quarks of Object-Oriented Development. In descending order of popularity, the 'quarks' are: Inheritance, Object, Class, Encapsulation, Method, Message Passing, Polymorphism, Abstraction.

[2] [http://en.wikipedia.org/wiki/Object-oriented\\_programming](http://en.wikipedia.org/wiki/Object-oriented_programming)