A Time Domain Beamformer Object Oriented Analysis and Design

Warren H. Axtell
Naval Undersea Warfare Center
Newport Division
New London, CT 06320

Abstract

This paper describes the analysis and design of a Time Domain Beamformer, using object modeling techniques, as a subset of a real-time sonar signal processing system. The goal of this work is to illustrate the relationship of portability and reuse to object oriented analysis. This task is part of an initiative to foster the application of fast paced, emerging commercial technology to the development and acquisition of affordable strategic systems at the Naval Undersea Warfare Center (NUWC).

1. Introduction

This paper describes the analysis and design of a time domain beamformer subset of a real-time sonar signal processing system using object oriented design techniques. This task is part of a continuing effort in the Efficient Approach to Systems Evolution (EASE)¹ project.

The goal of this paper is:
• to describe object oriented analysis and design concepts,
• to demonstrate the application of these concepts to the time domain beamformer process, and
• to illustrate the relationship between object oriented design and software portability and reuse.

The object oriented model described in this paper is a high level model. The next stage of this effort will be the development of a prototype implementation of this model in C++ language and the object oriented modeling of system control, signal processing functions and the display interface.

2. Object Oriented Design & Modeling Approach

A. Object Oriented Design Concepts

Object oriented design is a method of abstracting the real world application domain in an object model. This abstraction and is developed independent of programming languages is expressed entirely in abstract concepts. The abstract model consists of objects which are characterized by their properties and their behavior. In the object model the properties are expressed as attributes and the behavior as operations.

The first step in abstraction is to create a textual description of the problem or application. The textual description is analyzed by extracting nouns, verbs and descriptors from the text. The nouns are objects, the verbs are operations and the descriptors are attributes. From this list of objects, operations and attributes the object model is built.
The objects represent abstractions of real-world entities within the application, encapsulate operations and attributes, and have relationships to each other. Objects may have communication relationships, aggregation relationships or inheritance relationships.

The object oriented model should be able to be implemented in any language which has paradigms for classes, i.e., C++, Pascal, Ada. The object oriented design methods used for this task are based on Rumbaugh's Object Modeling Technique as described in Object Modeling and Design. Other object oriented design methods, such as Booch's, for example, are similar in method and differ primarily in their graphical notation, and the language emphasized for the design implementation. Although the design is not language dependent, Rumbaugh emphasizes C++ and Booch emphasizes Ada.

Rumbaugh's Object Modeling Technique includes the development of three views of the application domain which are described by three models: the Object Model, the Functional Model and the Dynamic Model.

1) Object Model
The Object Model is a view of the application domain in which the objects are discrete, distinguishable entities in the application domain. The Object Model describes the structure of objects in a system including their identity, relationships to other objects, their attributes, and their operations. The object model provides the structure into which the dynamic and functional models can be placed.

In object oriented design the Object Model is emphasized and is organized around real-world and conceptual objects. This can be compared to structured analysis and design methods in which the functional model is dominant and the design is structured around the functionality.

Objects with the same attributes and behavior are grouped into object classes. Classes are metadata models of object instances. Metadata refers to data that describes other data. A particular object with specific data is an instance of an object class. Instantiation refers to the implementation of an object class in a specific object with actual data.

The object encapsulates its data structure as attributes and its behavior as operations. Data within an object is accessible to other objects by calling functions within that object, thus making the objects loosely coupled. Internal data is hidden and accessed via parameter lists in function calls.

2) Functional Model
The Functional Model is used to describe transformation of values in the application. Functional models are represented in data flow diagrams. Functions in this model correspond to actions in the dynamic model and operations in the object model.

3) Dynamic Model
The Dynamic Model is used to describe aspects in an application that involve timing and sequencing of operations. The Dynamic Model describes system control, sequence of events, and states. Actions in the
Figure 3. Time Domain Beamformer Object Model
Dynamic Model correspond to functions in the Functional Model. Events in the Dynamic Model correspond to operations on Objects in the Object Model.

B. Object Oriented Design Tool

The Object Modeling Tool (OM Tool) graphical design software package was used for this project. OM Tool is a graphical design tool which provides a framework for designing object, functional and dynamic models for an application. This software package also generates C++ header code and code skeletons with function calls and input/output parameters.

Other software packages were investigated but OM Tool was chosen because of:
- the benefit of using the same graphical notation throughout the whole life-cycle of the analysis, design and implementation process and
- the ability to produce object, dynamic and functional models showing a more complete abstraction of the application domain.

The author's previous training in the Rumbaugh modeling method, its graphical representations, and the low cost of the initial software package contributed to this choice.

C. Object Oriented Design Notation

The design notation provided by OM Tool was used for this task. A summary of the graphical notation can be found in Object Modeling and Design. The implementation of this notation in the Object Oriented Models is described in the sections below for each model.

3. Time Domain Beamformer Object Oriented Models

A. Object Model

a. System Level

Figure 2 is a system level object model showing the signal processing system as an aggregate of its constituent parts i.e. the array, receiver, beamformer, detection processor, tracking processor and display processor.

The beamformer object is broken down further into smaller objects and the beamformer inputs from the receiver and outputs to detection processing are modeled as viewed from the beamformer.

b. Time Domain Beamformer

Figure 3 is an object model of a time domain beamformer with the functionality described below.

The time domain linear beamforming process being modeled is a two stage process. In this model the beams are formed from linear sensor data which is input from a two dimensional array with rows of sensors. The following procedure is performed for all beam conical directions:

- the sensor data is read from an input data store in a format corresponding to the coarse time delay and beam ID for the beam being formed,
- shading and deleting is applied to the sensor data,
- data with the same fine time delay is partially summed, fine time delay interpolation is performed and
- the data is summed to form row beams.

In stage two
- vertical shading is applied to the row beam data, and
- the shaded data is summed to form the complete beam data value. The beam data value has a beam ID which corresponds to a conical beam direction.

The rationale for this object design is to encapsulate the operations and data operated on in object classes which have identity in the actual system. The objects are conceptual abstractions of real-world entities which exist in the beamformer application. The objects shown in figure 3 are Sensor Array System, Beamformer, Beam, Array Row, Sensor, and Sensor Data Sample. This choice of objects is based primarily on viewing the beamformer application from the top level to the atomic level with each object having an aggregation relationship with the object above. In reducing an object into its aggregate parts the atomic level is the lowest level in which the aggregate parts are still meaningful objects. In this case the sensor is the lowest meaningful aggregate part in breaking down the objects. For example, sensor is part of array row, array row outputs are combined to form beams, etc. (Italics are used to indicate objects.)

Objects will be instantiated many times from each class, for example beam will be instantiated for each beam formed, array row will be instantiated for each row in each beam, etc.

The following section describes the object classes used in this model, the data held and the functions performed by each object.

1. Object Class Beamformer

Beamformer will be instantiated to perform the beamforming which is characteristic of the sensor array system of which it is a part. Beamformer will initiate the beamforming processes, select the beams to be formed, call operations in beam which will return beam data, and make the resulting beam data available for output.

Beamformer has the following attributes:
- beam data records which hold the beam data formed for each beam selected. Each beam data record contains an identifying beam ID and a beam data value.

Beamformer performs the following operations:
- initiates the beam processing for each beam, and
- outputs the beam data to the next sensor array system processing stage.

2. Object Class Beam

Beam will be instantiated by a call from beamformer for each beam that is formed, i.e. there will
be a separate instance of Object Beam for each beam formed, execute operations encapsulated in array row to obtain row beam data, then combine the row beam data from all rows to form the complete beam. The complete beam data is then returned to beamformer.

Beam will hold the beam data for a particular beam identified by a beam ID, create beam data and sum row beam data. Object beam will be instantiated for each beam being processed. Each beam is made up of row beams. Row beam data used to create beam data is obtained by calling functions in object array row.

Beam has the following attributes:
- Beam ID
- Beam data value

Beam performs the following operations:
- Form beam data
- Sum row beam data

(3) Object Class Array Row
Array Row will be instantiated for each row in the particular beam of which the row is a constituent part, perform the input of sensor data sample from Object Sensor, the summation of data samples in a row, and
application of vertical shading to row sum. The row sum data is then returned to Object Beam for summation.

*Array Row* has the following attributes:
- Row number
- Row beam data record
*Array Row* performs the following operations:
- Form row beam
- Sum sensor data
- Apply vertical shading to sensor data sum

(4) **Object Class Sensor**
Object Sensor holds the interpolated sensor data samples, sensor ID, and delete/not-delete data. Sensor is instantiated for each sensor in the array and obtains data from a data store of sequential time data for each sensor. This object inputs two sequential data samples for each sensor, interpolates them to form one data value and returns the value to Object Array Row for summation in the Row Beam.

*Sensor* has the following attributes:
- Sensor Data Value
- Sensor ID
- Shading coefficient data
- Delete/not-delete data
*Sensor* performs the following operations:
- Get sensor data sample
- Apply shading and deleting
- Store Data Sample
- Do fine time delay interpolation
- Output sensor data

(5) **Object Class Sensor Data Sample**
Object Sensor data sample represents each individual data sample in the data store. Each data sample has a value, a sensor ID, and a coarse time delay value represented by its sequential position in the input data stream. A call to sensor data sample will return the particular data sample required to create a particular row in a particular beam.

*Sensor Data Sample* has the following attributes:
- Sensor Data Value
- Sensor ID
- Coarse Time Delay Value
*Sensor Data Sample* performs the following operations:
- Get sensor data

c. **C Code Skeleton Generated from Object Oriented Model**
A C++ code skeleton can be generated by the OM Tool Software directly from this object model.

**B. Functional Model**
The Functional Model describes the data transformations in the system. In the data flow diagrams, ovals represent processes, lines represent data flows, double horizontal lines represent data stores, and rectangles represent actors. Actors are objects that drive the data flow by producing or consuming values.

The Functional Model, shown in figures 4 and 5, show the data processing which takes place in the Beamforming Model. The figures are data flow diagrams (DFD) showing the processes which take place from sensor data input through creation of complete beam data. These DFD's show a single stream of processing.

Data processing for multiple sensors and multiple beams will occur through parallel streams. The formation of beams involves concurrent parallel processes. For each beam formed, the formation of its constituent row beams may be performed as concurrent processes. For each row beam formed, the input, shading and interpolation of data samples from each sensor in the row may be performed as concurrent processes.

**C. Dynamic Model**
The Dynamic Model describes those aspects of the system that have to do with time, changes, and control. In the dynamic model the main concepts are events and states. Events are external stimuli in the system and are represented as lines in the model. States are values of objects and are represented as ellipses in the model. The time domain beamformer is a procedure-driven sequential system, therefore the dynamic model is less important than the object and functional models. The external stimuli is data input; the timing aspect requiring control is the rate of data input. The only external controls required start processing and stop processing.

The Dynamic Model in this application will deal with sensor data input and initiation of processing. Figure 6 shows the dynamic model state diagram for controlling
4. Object Oriented Model Portability/Transportability

The object oriented model encapsulates data and operations in objects which are highly cohesive and loosely coupled. These features enhance the portability of the model to multiple platforms and transportability of the object classes to multiple applications.

The object model is an abstraction of the application domain. The model is a pictorial abstraction that provides the flexibility to implement the application in any language which provides appropriate paradigms for object oriented programming. The languages would include C++, Pascal, Ada, Eiffel, Smalltalk and others. OM Tool creates a C++ code skeleton including function declarations. The implementation of functions within an object is hidden from other objects therefore maintenance or modification of functions within an object will not impact the overall system design.

Objects which are well defined and loosely coupled are reusable in other applications. This beamforming application was designed with a large hull array in mind, however, the objects could be reused in an application of a towed array with a single line of sensors. The same objects and operations would apply. There would either be a single row or if the array were processed in segments, each segment would be a row with appropriate shading. Then the segments would be summed.

Object models and resulting implementations are more readily reusable in other applications in comparison to traditional structured analysis and design models because the functional implementation of the traditional models produce closely coupled software modules which are interdependent. A major change to one module will require a redesign of the whole system because the functions dependent on each other. In contrast object design uses loose coupling and information hiding to isolate the functions with objects and provide interfaces to the objects to access the data and functions. The software interface which is visible to other objects remains constant while the implementation and/or data within the object may be modified without impacting the rest of the design.

Consequently this object oriented design has a potential reuse for other acoustic transducer configurations and other sensor array systems including acoustic, photonic and radar systems.

An additional benefit inherent in object oriented analysis and design is the consistency of modeling methods throughout the life-cycle of analysis, design, implementation, testing and maintenance of an application. The same object oriented concepts are carried throughout the whole life-cycle including any required redesign during the maintenance phase. This is due to the natural result abstracting the real-world view of the system. In structured analysis and design using the waterfall method the modeling techniques change in each stage of the development. In the object oriented development life-cycle the same object model is refined from one stage to the next.
5. **Summary**

**A. Issues to be resolved**

**Scope of application**

The beamforming process being modeled includes a large number of sensor inputs, high input data rates, large data storage requirements, and a large number of beams output. To process these large quantities of data will require concurrent processing of data. The creation of each beam is a concurrent process stream. Within each beam the creation of each row beam is a concurrent process stream.

A means of controlling the concurrent processing must be developed. Concurrency requires a semaphore to initiate multiple processing streams and to synchronize the streams at the end of the processing. In the Ada programming language this is performed by the task specifications with task entry and rendezvous operations. There are shell programs which provide semaphores for concurrent parallel programming with the C language used for implementations.

**System design**

Design of the overall system is a separate issue which will require a module design. This will require an architecture design for control of system level objects.

**Data input**

The input of large quantities of sensor data will be dependent on the operating system implementation. Data input will require a separate software module with a separate object model which can be called by the beamformer module for data access.

**B. Future Direction**

A scaled down implementation of object model in an object oriented language for use as prototype and benchmark will be developed. After completion of a software specification from this design model, implementation will be in the C++ language. The next phase of Object Model development for the Signal Processing System is modeling the system control, detection and display portion of the processing.

**C. Goals**

The experience of most software design engineers is with using structured analysis and design techniques for system design. Our ultimate goal is to demonstrate that object oriented design will produce a software product that is more easily understood because it models the real world as an abstraction of the application, that is reusable, that is extendible to related applications, and that is portable for implementation on many platforms with any object oriented language.

References

3. G. Booch, Object-Oriented Design with Applications, Benjamin/Cummings, Redwood City, CA, 1991
4. OMTool User Guide, Martin Marietta Corp, King of Prussia, PA, 1993