2007 International IEEE Symposium on Precision Clock Synchronization (ISPCS)
for Measurement, Control and Communication
Vienna, Austria, October 1-3, 2007.

Object-oriented Model for IEEE 1588 Standard

Kang Lee
National Institute of Standards and Technology
100 Bureau Drive, MS# 8220
Gaithersburg, Maryland USA 20899-8220
E-Mail: kang.lee@nist.gov

Eugene Song
National Institute of Standards and Technology
100 Bureau Drive, MS# 8220
Gaithersburg, Maryland USA 20899-8220
E-Mail: ysong@nist.gov

Abstract

The IEEE 1588 standard specifies a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, and distributed objects. The Unified Modeling Language (UML) is a powerful tool for object-oriented modeling, design, and development of complex distributed systems. This paper describes an object-oriented model for the IEEE 1588 standard-v2, which has been developed using UML tool at National Institute of Standards and Technology (NIST). This model consists of the data types, datasets, entities, and devices of IEEE 1588 standard-v2. The model has been used to produce C++ source codes, and create C++ libraries for the IEEE 1588 standard-v2. With the help of this object model, the development time of IEEE 1588-basd distributed measurement and control applications can be reduced dramatically.

Keywords: Time Synchronization, IEEE 1588, Object-oriented Model, Precision Time Protocol

1. Introduction

The IEEE 1588 standard defines a precision time protocol (PTP) enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects [1]. The protocol is applicable to systems communicating via packet networks. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources. This IEEE 1588 protocol is applicable to distributed measurement and control systems consisting of one or more nodes, communicating over a network. Nodes are modeled as containing a real-time clock that may be used by applications within the node for various purposes such as generating timestamps for data or ordering events managed by the node.

Object-oriented development methodology is a robust and flexible software development approach because it provides a better way to organize software allowing the developer to build better, scalable, and more complex software with less effort [2]. UML (Unified Model Language) is a modeling language for supporting object-oriented modeling, design, and development by expressing the constructs and the relationships of the components of a complex distributed system. It combines the methods of Booch, Rumbaugh, and Jacobson [3-5]. The Object Management Group (OMG) accepted UML as its standard for modeling object-oriented systems in 1997 [6]. The new SEMI standard defines a clock object, which allows a host system or the factory to query about the equipment or application time synchronization quality and status. The attributes of the clock object are based on selected attributes defined in Network Time Protocol (NTP), and used to access the clock object and to enable or disable time synchronization through the factory network [7-8]. So far, no object-oriented model of the IEEE 1588 standard exists today. NIST researchers are interested in developing an object-oriented model for the IEEE 1588 standard in order to reduce development time for IEEE 1588 applications.

2. Object-Oriented Model for IEEE 1588 Standard

In an object-oriented design approach, the designer uses classes to define data types. A class may add functionality to an existing type or extend the current type capabilities to create a completely new type, a derived type. The IEEE 1588 data types, datasets entities, and devices can be modeled based on object-oriented model approach using the UML tool.

2.1. Object-oriented Data Object Model of IEEE 1588 Data Types

The IEEE 1588 data types can be classified into primitive data types and derived data types (or structured data type). The primitive data types include Boolean, Integers (signed and unsigned 8 bits, 16 bits, 32bits, and 64 bits), and Octet (an 8 bit unsigned char). The derived data types of the IEEE 1588 shall be derived from these primitive data types. The derived data type includes arrays of the primitives, structs, and enumerations. The following describes the object-oriented data model of the IEEE 1588 standard-v2 in UML tool.

2.1.1. Mapping Primitive Data Types of IEEE 1588 to C++

The primitive data types of IEEE 1588 include Boolean, Integers (signed and unsigned 8 bits, 16 bits, 32 bits and 64 bits), and Octet (an 8 bit unsigned char). There are exactly mapping the primitive types in C++ language, referred to as bool, char, short, int, long. So the IEEE 1588 primitive data types can be directly mapped into

C++ primitive data types. For example, we use typedef to define UInteger8 as unsigned char in C++. Table 1 shows the mapping of IEEE 1588 primitive data types to C++.

Table 1 Mapping primitive data types of IEEE 1588 to C++

IEEE 1588 primitive data types	C++ Primitive Data Types	Definition
Boolean	bool	typedef bool Boolean
Octet	Unsigned char	typdef unsigned char UInteger8
Integer8	char	typedef char Integer8
Integer16	short	typedef short Integer16
Integer32	int	typedef int Integer32
Integer64	long	typedef long Integer64
UInteger8	unsigned char	typdef unsigned char UInteger8
UInteger16	unsigned short	typdef unsigned short UInteger16
UInteger32	unsigned int	typdef unsigned int UInteger32
UInteger64	Unsigned long	typdef unsigned long UInteger64

2.1.2 Definitions of Derived Data Types in C++

The derived data types of the IEEE 1588 include arrays of primitive data types, enumerations, and structs. C++ is an object-oriented programming language, so all derived data types for IEEE 1588 can be defined as classes in C++. Table 2 shows the mapping of derived data types of IEEE 1588 to C++. We define these structured data types as classes.

Table 2 Mapping derived data types to C++

IEEE 1588 Derived Data Types	C++
array	class
Enumeration	enum
typedef	subclass or attributes of class
struct	class

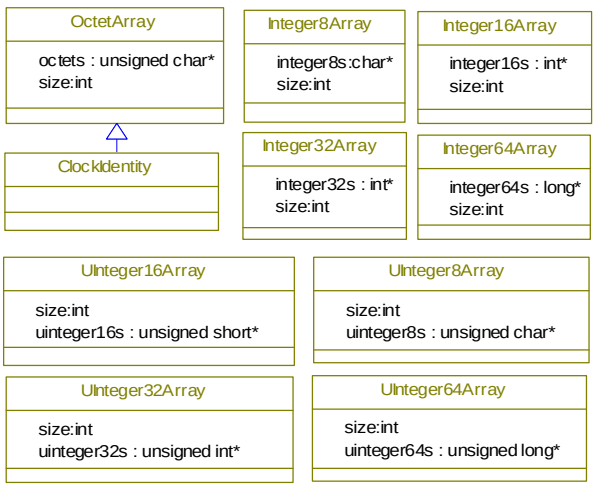

Figure 1. Arrays of primitive data types of IEEE 1588.

2.1.2.1. Definitions of Array of Primitive Data Types

All the arrays of primitive types can be defined as a class in C++. For example, Integer8Array can be defined as a class and Integer8 can be mapped into a char in C++. So Integer8Array can be defined as a class with attributes integer8s (array of char) and a size of the array.

Figure 1 shows the class definitions of all arrays of primitive data types of the IEEE 1588 standard. These classes include OctetArray, Integer8Array, Integer16Array, Integer32Array, Integer64Array, UInteger8Array, UInteger16Array, UInteger32Array, and UInteger64Array.

2.1.2.2. Subclass Definition of Typedef

The typedef of C++ can be used to define user data types based on the existing data types. It is normally based on primitive data types. When the existing data types are classes, the typedef can be used to define a new data type by deriving a sub-class from the existing classes. For example, `typedef Octet[8] ClockIdentity`, Octet[] can be defined as a class OctetArray, which is array of Octet (unsigned char), so clockIdentity is a subclass of OctetArray. Figure 1 shows the inheritance relationship between ClockIdentity and OctetArray.

2.1.2.3. Definitions of Struct Data Types

The struct data type of IEEE 1588 can be mapped to a C++ class with the same name. The members of struct can be mapped to the attributes of the class. The attributes of a class can also be represented through the associations. The struct data types of IEEE 1588 include TimeInterval, Timestamp, PortIdentity, PortAddress, ClockQuality, TLV, PTPText, and FaultRecord. Figure 2 shows class definitions of these struct data types.

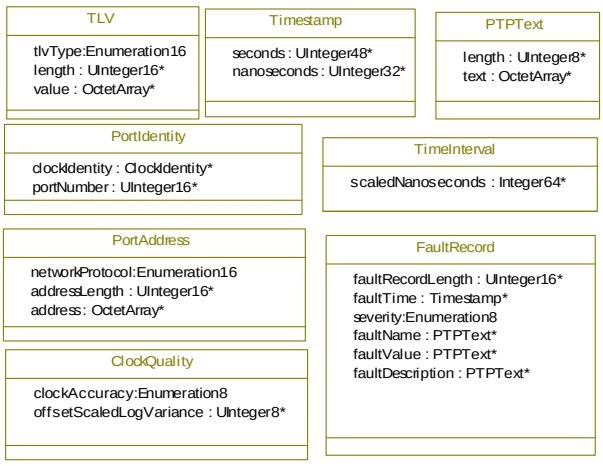

Figure 2. Classes of struct types.

There are some data types defined in IEEE 1588-v2, which are not based on smallest native data types (byte, or char), such as UInteger4, UInteger48, and Enumeration4. Although the smallest native data type of C++ is char, the language does enable us to manipulate bits directly without resorting to assembly programming or

inefficient libraries. A bit-field is a data member of a struct or a class which contains one or more bits. The underlying type can be signed char, short, int, long, unsigned counterparts. For example, Nibble, UInteger4, and UInteger48 can be defined as struct in the following.

```
typedef struct Nibble {
     unsigned char nibble:4;
} Nibble;

typedef struct UInteger4 {
     unsigned char uinteger4:4;
} UInteger4;

typedef struct UInteger48 {
   unsigned short first:16;
   unsigned short second:16;
   unsigned short third:16;
} UInteger48;
```

2.1.2.4. Definitions of Enumeration

The enumerations of IEEE 1588 can be directly mapped into enum of C++. These enumerations include Enumeration4, Enumeration8, Enumeration16, ClockAccuracy, ClockType, TimeSource, PTPState, DelayMechanism, MessageType, Control, Action, InitializationKey, FaultLog, ManagementErrorID, TrustState, ChallengeType, PTPTimeScale, TimeAccuracy, PortState, Version, MessageClass, MessageTransmissionIntervals, GeneralMessage, EventMessage, PTPNetworkProtocol, and AddressType. For example, the enumeration MessageType can be defined in the following.

```
enum MessageType
{
  SYNC,
  DELAY_REQ,
  PDELAY_REQ,
  PDELAY_RESP,
  FOLLOW_UP,
  DELAY_RESP,
  PDELAY_RESP_FOLLOW_UP,
  ANNOUNCE,
  SIGNALING,
  MANAGEMENT
};
```

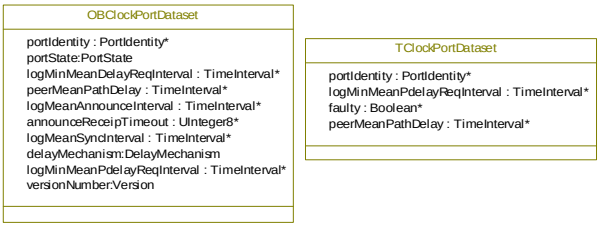

Figure 3. Port dataset of clock.

2.2. Definitions of IEEE 1588 Datasets

2.2.1. Port Dataset

There are two kinds of port datasets defined in IEEE 1588. They are ordinary and boundary clock port data set and transparent clock port dataset. Figure 3 shows PortDataset of ordinary and boundary clock, and PortDataset of transparent clock. The OBClockPortDataset in Figure 3 is a port dataset of ordinary and boundary clocks. TClockPortDataset in Figure 3 is port dataset of transparent clock.

2.2.2. Ordinary and Boundary Clock Dataset

The ordinary and boudary clocks have four datasets: DefaultDataset, CurrentDataset, TimePropertyDataset, and ParentDataset. Figure 4 shows ordinary and boundary clock dataset, which includes DefaultDataset, CurrentDataset, TimePropertyDataset, and ParentDataset. Each dataset can be defined as a class.

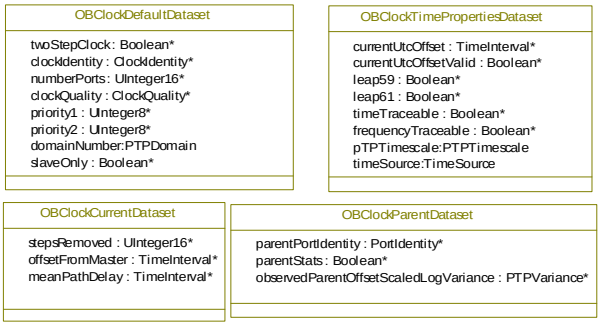

Figure 4. Ordinary and boundary clock datasets.

2.2.3. Transparent Clock Dataset

The transparent clock has two datasets: default dataset and current dataset. Figure 5 shows the object model of transparent clock dataset,. TClockDefaultDataset is the default dataset; TClockCurrentDataset is the current dataset.

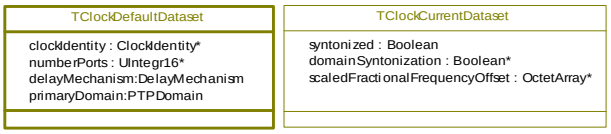

Figure 5. Transparent clock dataset.

2.3. Definitions of IEEE 1588 Entities

The IEEE 1588 PTP entities include PTPMessage, PTPTimeouts, PTPPort, PTPVariance, P2PResidenceTimeBridge, E2EResidnceTimeBridge and FlagIndicator. These entities can be defined as classes shown in Figure 6. The PTP port is a logical access point of a PTP clock for PTP communications to the communications network. Each port on a PTP ordinary, boundary , and transparent clock is modeled as supporting two interfaces, event and general. The event interface is used to send and receive event messages, which are timestamped by the timestamp generation block based on the value of the local clock. The general interface is used to send and receive general messages.

Figure 6 shows the definition of PTPPort class. The PTP Port has one PortIdentity, one PortAddress, one EventInterface (PTPMessage), one GeneralInterface (PTPMessage), one TimeStamp, one PortState, one Path DelayMechanism, and one Version. The state machine

sign is shown in upper-right corner of class diagram of PTPTPort. The PTPPort C++ source code (`PTPPort.h` and `PTPPort.cpp`) generated from this model is described in the following.

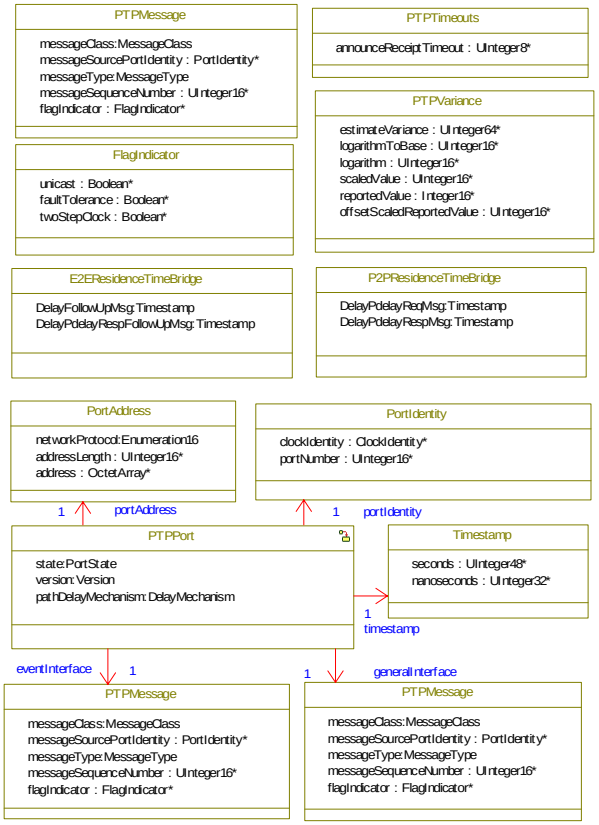

Figure 6. Class of PTP entities.

// PTPPort.h

```
class PTPPort : public OMReactive {

PTPPort(OMThread* p_thread = OMDefaultThread);
PTPPort(const PTPPort& a, OMThread* p_thread =
OMDefaultThread);
~PTPPort();

public :
const PTPPort& operator=(const PTPPort& a);

//    Attributes
protected :
    DelayMechanism pathDelayMechanism;
    PortState state;
    Version version;

//Relations and components
protected :
    OBClockPortDataset* bClockPortDataset;
    PTPMessage* eventInterface;
    PTPMessage* generalInterface;
    PortAddress* portAddress;
    PortIdentity* portIdentity;
    Timestamp* timestamp;
};
```

// PTPPort.cpp

```
PTPPort::PTPPort(OMThread* p_thread) {
    setThread(p_thread, FALSE);
    bClockPortDataset = NULL;
    eventInterface = NULL;
    generalInterface = NULL;
    portAddress = NULL;
    portIdentity = NULL;
    timestamp = NULL;
    initStatechart();

}

PTPPort::PTPPort(const PTPPort& a, OMThread*
p_thread) {
    timestamp = NULL;
    portIdentity = NULL;
    portAddress = NULL;
    generalInterface = NULL;
    eventInterface = NULL;
    bClockPortDataset = NULL;
    setThread(p_thread, FALSE);
    initStatechart();
    (PTPPort&)*this=(PTPPort&)a;
}

PTPPort::~PTPPort() {
    cleanUpRelations();
}

const PTPPort& PTPPort::operator=(const PTPPort&
a) {
    (PTPPort&)*this=(PTPPort&)a;
    return *this;
}
```

2.4. Object Modeling of the IEEE 1588 Devices

A clock is capable of providing a measurement of the passage of time since a defined epoch. A PTP clock is a clock that participates in the PTP protocol. Different clocks have different datasets. There are five types of PTP devices: LocalClock, Ordinary clock, Boundary clock, End-to-end transparent clock, Peer-to-peer transparent clock, and Management node. A LocalClock is a physical clock with a timestamp. All other PTP devices are identified by a clock identity attribute, which is included in PortIdentity.

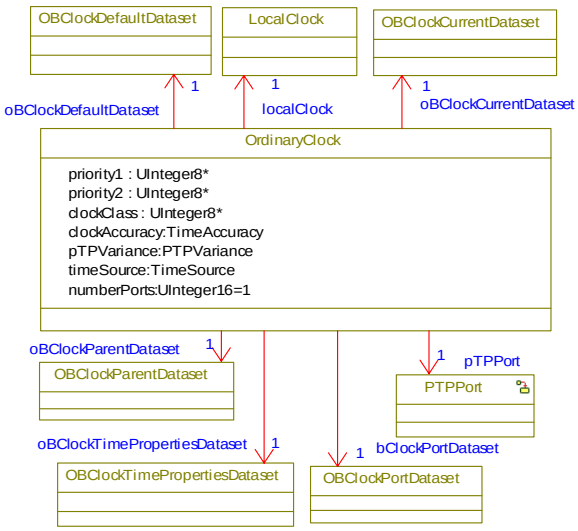

Figure 7. Object model of ordinary clock.

2.4.1. Object Model of IEEE 1588 Ordinary Clock

An ordinary clock communicates with the network via two logical interfaces based on a single physical port. The ordinary clock can be the grandmaster clock in a system or it can be a slave clock in the master-slave hierarchy. Figure 7 shows the object model of ordinary clock. Each ordinary clock has one LocalClock, one OBClockDefaultDataset, one OBClockCurrentDataset, one OBClockParentDataset, one TimePropertiesDataset, and only one OBClockPortDataset. The ordinary clocks are characterized by the attributes: priority1, priority2, clockClass, clockAccuracy, timeSource, pTPVariance and numberPorts. Each ordinary clock may have only one PTPPorts.

2.4.2. Object Model of IEEE 1588 Boundary Clock

A boundary clock translates the PTP protocol messages between regions implementing different transport and messaging protocols. A boundary clock typically has several physical ports with each physical port communicating with the network via two logical event and general interfaces. Each port of a boundary clock is like the port of an ordinary clock with the following exceptions. The clock data sets are common to all ports of the boundary clock. The local clock is common to all ports of the boundary clock.

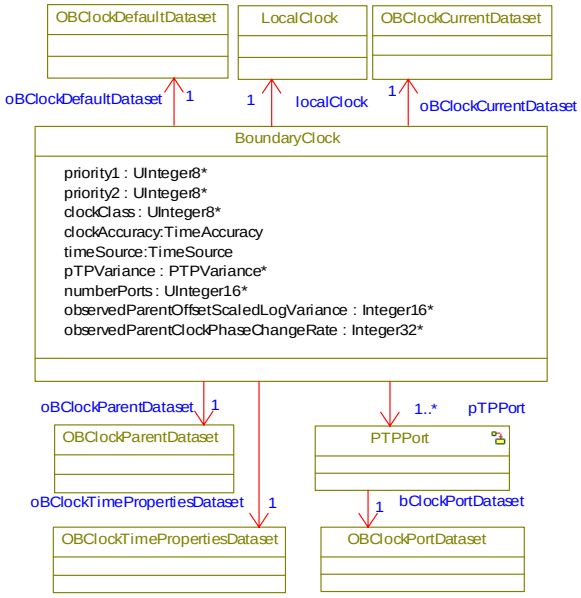

Figure 8. Object model of boundary clock.

Figure 8 shows the object model of boundary clock, which has one LocalClock, one OBClockDefaultDataset, one OBClockCurrentDataset, one OBClockParentDataset, one TimePropertiesDataset, and one or many PTPPorts. Each PTPPort has one OBClockPortDataset. The boundary clocks are characterized by the attributes: priority1, priority2, clockClass, clockAccuracy, timeSource, pTPVariance, and numberPorts.

2.4.3 Object Model of IEEE 1588 End-To-End Transparent Clock

A transparent clock translates the PTP protocol messages between regions implementing different transport and messaging protocols. The end-to-end transparent clock forwards all messages just as a normal switch, router, or repeater. However for PTP event messages, the residence time bridge measures the residence time of PTP event messages. These residence times are accumulated in a special field, the correction field, of the PTP event message or the associated follow up message.

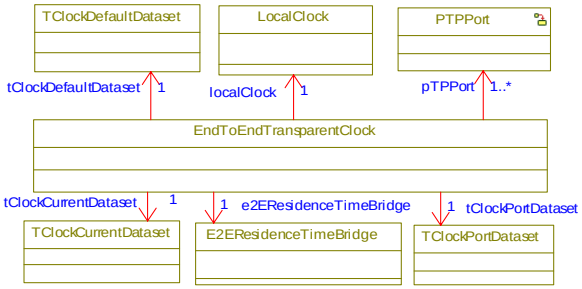

Figure 9. End-to-end transparent clock.

Figure 9 shows the object model of end-to-end transparent clock. Each end-to-end transparent clock has one LocalClock, one TClockDefaultDataset, one TClockCurrentDataset, one TClockPortDataset, one E2EResidenceTimeBridge, and one or many PTPPorts.

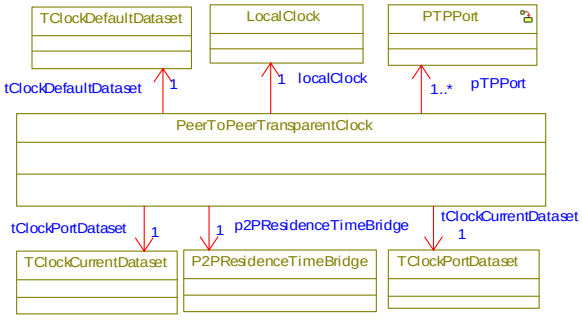

Figure 10. Peer-to-peer transparent clock.

2.4.4 Object Model of IEEE 1588 Peer-To-Peer Transparent clock

The peer-to-peer transparent clock differs from the end-to-end transparent clock in the way it corrects and handles the PTP timing messages. The peer-to-peer transparent clock can be associated with an ordinary clock in exactly the same way as an end-to-end transparent clock. The peer-to-peer transparent clock has an additional per port block. This block is used to compute the link delay between each port and a similarly equipped port on another node sharing the link, i.e., the link peer. The link peer will be in another clock supporting the peer delay mechanism since non-peer-to-peer devices are not expected between peer-to-peer transparent clocks. Figure

10 shows the object model of peer-to-peer transparent clock. Each peer-to-peer transparent clock has one LocalClock, one TClockDefaultDataset, one TClockCurrentDataset, TClockPortDataset, and one or many PTPPorts.

3. Library Building and Application of IEEE 1588

This model has been used to generate C++ source code, and successfully compiled and built a C++ library that includes a set of classes of the IEEE 1588 standard. Figure 11 shows the interface of library building of object-oriented model of the IEEE 1588 standard. IEEE 1588 application developers can use it to implement their IEEE 1588 applications based on the established library.

A PTP system is a distributed, networked system consisting of a combination of PTP and non-PTP devices. PTP devices include ordinary clocks, boundary clocks, transparent clocks, and management nodes. Devices in a PTP system communicate with each other via a communication network. The network may include translation devices between segments implementing different network communication protocols.

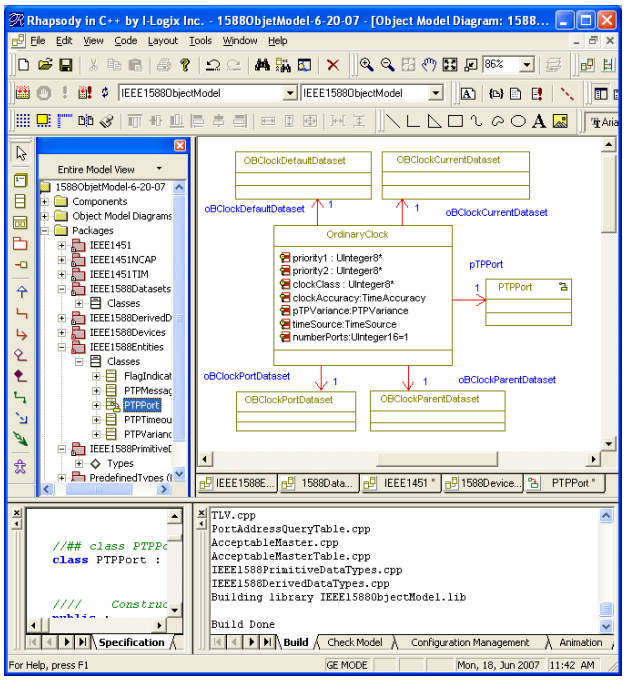

Figure 11. Interface of library building for IEEE 1588.

Within sensor networks, establishing the precise time that a particular observation or measurement is made is of crucial importance. For instance, when raw sensor data is fused from multiple sources it is imperative that the data was generated at the exact same moment as when the event occurred. It is important to synchronize the time of sensor nodes for sensor data fusion. It is possible to integrate the IEEE 1451 and the IEEE 1588 standards for distributed and synchronized measurement and control applications using this model. An IEEE 1451 Network Capable Application Processor (NCAP) uses a boundary (master) clock with one or more PTP ports; An IEEE 1451 Transducer Interface Module (TIM) uses an ordinary (slave) clock with one port. So a number of TIMs can synchronize with the NCAP to implement synchronized and distributed measurement and control.

4. Conclusion

We have developed the object-oriented model for the IEEE 1588 standard-v2. The model has been used to produce C++ source codes, and create a C++ library for the IEEE 1588 standard. Using this object-oriented model, the development time of IEEE 1588 applications can be reduced dramatically.

Our future work is to develop the reference implementation of the IEEE 1588 standard based on this object-oriented model, and to integrate with the IEEE 1451 standard to do synchronized measurement and control.

** Commercial equipment and software, many of which are either registered or trademarked, are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

References:

[1] IEEE P1588D2.1, Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2007.

[2] Lee, Kang, Song, Eugene Y. UML Model for the IEEE 1451.1 Standard, IEEE Instrumentation and Measurement Technology Conference, Vail, CO, May 2003.

[3] Fowler, Martin., Scott, Kendall. UML Distilled (second edition) - A brief guide to the standard object modeling language, Addison-Wesley, 1999.

[4] Douglass, Bruce. Real-time Object-oriented Model (second edition): Developing efficient objects for embedded systems, Addison-Wesley, 1999.

[5] James, Rumbaugh., Ivar, Jacobson., Grady, Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[6] UML, http://www.omg.org/uml/ [last updated; January 02, 2007]

[7] Gino Crispieri, Software Timing Synchronization, http://www.future-fab.com/documents.asp?d_ID=4405 [last updated: 7/9/2007]

[8] NTP, http://www.ntp.org/ [last updated: August 06, 2007.]