IEC 61850 Prototype Design

Suhas Aggarwal
Indian Institute of Technology Guwahati
suhasagg@gmail.com

ABSTRACT

In this paper, we discuss information model of IEC 61850 protocol. We give the prototype design of information model of IEC 61850 and study record accessing times. We suggest use of real time main memory database model for IEC 61850. We also use TLSF memory allocator for assigning memory to records in IEC 61850 prototype. We suggest a back up scheme for database for fast crash recovery. Finally we study transaction processing and security of database.

1.Introduction

The recent international standard IEC 61850 proposes a unified solution of the communication aspect of substation automation. IEC 61850 consists of a data model and service model. In this paper we will focus on information model.

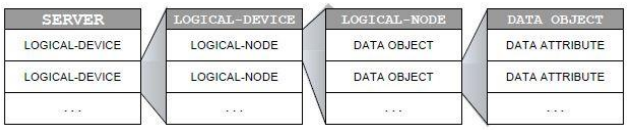

Figure 1: Hierarchy of the IEC 61850 data model

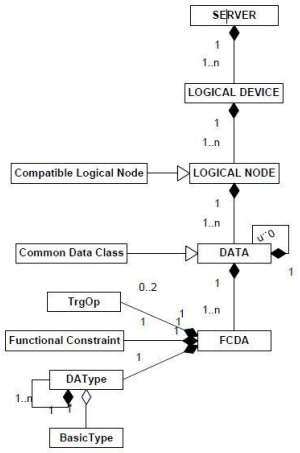

Figure 2: The data model of the IEC 61850

The hierarchical data model defined in the IEC 61850 is depicted in Figure 1 and Figure 2. Server is the topmost component in this hierarchy. It serves as the joint point of physical devices and logical objects. Theoretically one IED[2] may host one or more server instances, but in practice usually only one server instance runs in an IED. A server instance is basically a program running in an IED, which shares the same meaning with other servers like FTP server etc. Each server hosts several files or logical devices. Clients can manipulate files in the server like talking to a FTP server, which is usually used as a means to upload/update the configuration file of an IED. A logical device is the logical correspondence of a physical device. It is basically a group of logical nodes performing similar functions. Functions supported by an IED are conceptually represented by a collection of primitive, atomic functional building blocks called logical nodes. Besides the regular logical nodes for functions, the standard also requires every logical device have two specific logical nodes: Logical Node Zero (LN0) and LPHD, which correspond to the logical device and the physical device,alternatively. Besides holding status information of the logical device, LN0 also provides additional functions like setting-group control, GSE control, sampled value control etc. Data exchanged between logical nodes are modeled as data objects. A logical node usually contains several data objects. Each data object is an instance of the DATA class and has a common data class type. Similar to the concept of objects in most object-oriented programming languages, a data object consists of many data attributes, which are instances of data attributes of the corresponding common data class. Data attributes are typed and restricted by some functional constraints. Instead of grouping data attributes by data objects, functional constraints provide a way to organize all the data attributes in a logical node by functions. Types of data attributes can be either basic or composite. Basic types are primitive types in many programming languages, whereas composite types are composition of a collection of primitive types or composite types. Despite data objects, the IEC 61850 standard provides the concept of data set as another ways to manage and exchange a group of data attributes. Members of a data set can be data objects or data attributes.

2. Information Model Database design

```
struct server
{
*logical_device[20];
}

struct logical_device
{
*logical_node[20];
}

struct logical_node
{
*data[20];
}

struct data
{
*data_attribute[20];
}
```

Allocate memory to server record dynamically using malloc. Memory allocated to 20 logical device pointers (total 80 bytes, 4 bytes for each pointer). As logical devices are being initialized, allocate memory to logical device records using malloc(memory allocated to entire logical device record), memory allocated to 20 logical node pointers (total 80 bytes,4 bytes for each pointer). As logical nodes are being initialised. allocate memory to logical node records using malloc(memory allocated to entire logical node record), memory allocated to 20 data pointers (total 80 bytes,4 bytes for each pointer). As data is being initialised allocate memory to data records using malloc(memory allocated to entire data record), memory allocated to 20 data attribute pointers (total 80 bytes,4 bytes for each pointer). Finally as data attributes are being intialized, allocate memory to data attribute records within a data using malloc.Involves memory allocation to fields namely name(string type), functional constraint (string type), trigger option (string type), value (depending on type field, int, float, string or composite type like analogue value (memory allocated accordingly)

2.1 Design Feature

Memory is allocated to each and every individual record as it is being created. For e.g - during initialization of server, memory is allocated to 20 logical device pointers (needing only 32 bits to store the address) not records. Later when individual logical device records are initialized, memory is allocated to entire logical device record containing 20 logical node pointers, this trend goes down the hierarchy.

2.2 Record Indexing and Accessing times

In order to access a logical device record within the server - 2 address reads are required one for reading the address of pointer to server object to find address of server record and one for reading address which contains server record which contain 20 pointers to logical device records.

To access logical device record-
Specify logical device no. (this no. serves as a key for accessing the record). Get address of pointer to logical device record O(1) (pointer address calculated directly => array [logical device no.]. Retrieve the logical device record via pointer (2 address reads are required, one corresponding to address of pointer and one corresponding to address of record present in the pointer), total overhead is summation of all the overheads mentioned above (2 address reads corresponding to server object + calculation of address of appropriate pointer + 2 address reads, one corresponding to address of pointer and one corresponding to address of record present in the pointer)

To access logical node record-
Specify logical node no.. Get address of pointer to logical node record present within logical device record (total overhead =>above calculated overhead + O(1) (pointer address calculated directly => array [logical node no.])). Retrieve the logical node record via pointer (2 address reads are required , one corresponding to address of pointer and one corresponding to address of record present in the pointer), total overhead is summation of all the overheads mentioned above .

To access data record-
Specify data no.. Get address of pointer to data record present within logical node record (total overhead =>above calculated overhead + O(1) (pointer address calculated directly => array [data no.])). Retrieve the data record via pointer (2 address reads are required, one corresponding to address of pointer and one corresponding to address of record present in the pointer), total overhead is summation of all the overheads mentioned above .

To access data attribute records present within the data-
Specify data attribute no.. Get address of pointer to data attribute record present within data record (total overhead =>above calculated overhead + O(1) (pointer address calculated directly => array[data attribute no.])). Retrieve the data attribute record via pointer (2 address reads are required, one corresponding to address of pointer and one corresponding to address of record present in the pointer, within the record, further pointers to various fields of data attributes will be present, 2 address reads will be required, one for reading the address of pointer and one for reading the field value whose address is contained in the pointer)

Alternate record indexing. Use of dynamic hashing.-
This is a slight variant of linear hashing. It may be useful for indexing the records at data and data attribute level where one may wish to access these records by specifying the individual data names and data attribute names. data name or data attribute name=>Unique number encoding. Hash(unique number encoding)=Appropriate bucket no.A sample set up - Initially hash table will contain 4 buckets. Bucket capacity =2
A sample set up

```
Bucket0  - data01 -02
   ||
Bucket1  - 11 -12
   ||
Bucket2  - data21 - data22
   ||
Bucket3  - 31 -32
```

Suppose an entry is to be inserted in bucket2 further which is full, linear hashing will split bucket0 first and redistribute the entries within bucket 00 and bucket 10 on addition of first overflow page in bucket2. On further addition of data entry in bucket 2, a second overflow page will be added (on assumption a page will contain a single data entry). This will result in splitting of bucket 1 into bucket 01 and bucket 11, each of capacity 2. Finally on addition of a data entry further to bucket 2, bucket 2 will be split and entries of bucket will be redistributed accordingly. I plan to split the bucket at the moment it has the tendency to overflow. Buckets won't be split in the order 0th, Ist and 2nd.As bucket 2 is about to overflow first, it will be split right at that moment without the addition of any overflow pages. This will require creation of buckets 10 and bucket 11 as well. But using above design feature, we can create just create pointer to bucket 10 and bucket 11 rather than creating a bucket record having a capacity of 2 data entries. Now as bucket 12 is to be filled, we can allocate memory to structure of bucket 12 having the space of 2 data entries. This will save unnecessary memory allocation and redistribution of values during record insertions. When bucket 0 or 1 is about to overflow, as it split pointers are already present, we can simply allocate memory to the pointer to split bucket structures and redistribute the entries.

```
struct hash_table
 {
  vector *bucket[4];
 }
struct bucket
{
*key;
*data[2];
}
```

One possible implementation can be by using vector of buckets which grows dynamically using pushback( ) function. In the above case pushback will be called 3 times. 3 pointer to bucket structures will be appended at the end of existing bucket vector.

2.3 Database type

A real time memory resident database. Database should reside in main memory all the times to save the disk latencies involved while fetching the data from the disk.

2.4 Memory Allocation of records

Refinement of memory allocation interface using TLSF dynamic memory allocator [8] especially suited for real time systems.
Comparison with other memory allocation algorithms-

	Allocation	Deallocation
First fit/Best fit	O(M/(2.N))	O(1)
Binary buddy	O(log(M/N))	O(log(M/N))
DL malloc	O(M/N)	O(1)
AVL	O(2.44.log(M/N)	O(4.32.log(M/N)
Half fit/TLSF	O(1)	O(1)

Base of logarithms is 2
M =Maximum memory size (Heap)
N =Largest allocated block

Table 1

2.5 Backup scheme for the database and fast crash recovery.[6]

Division of information into categories
1)Temporal
2)Persistent
Temporal region will consist of information which is frequently updated or changed such as data or data attribute records. Persistent region will consist of information which is almost static such as logical device, logical node records. Creating checkpoints to take efficient back up of data. Temporal region has very high checkpoint frequency. Persistent region has low checkpoint frequency. At checkpoints database image present in memory is copied to the disk. Data and data attribute records are changed pretty frequently so they are backed up at a much higher rate than persistent data such as logical node and device information. In case of a crash, database image corresponding to the latest checkpoint is loaded in the memory.

Reload threshold - When certain segments of database image is loaded in memory, database can start operating and processing transactions or service commands. This feature enables fast recovery.

3. Transaction processing

IEC 61850 server is a multi-client server . There should be assignment of priority to transactions. Server maintains a priority queue per client for processing transactions. Priority can be set on the basis of transaction deadlines. Also a queue can be maintained for transactions demanding operation on same data when issued by multiple clients. For assigning priority criteria can be transaction deadlines. This also require including a time stamp field in request and response data units of service interface. Also locking of data pages should be enabled. Say for eg- a client has issued the command setdatavalues (page whose contents are being edited should be locked, so that concurrent commands issued by other clients such as getdatavalues from that page should be blocked till the transaction is met ,this will save retrieval of improper datavalues(say from partially written or changed page)Also, condition checks operating on data should be paused while data setting is in process.

4. Security of database

Access control lists should be maintained. Implementation of views (access control feature present in IEC 61850). Based on authentication parameters which clients pass while setting up a connection with the server, they are granted a certain view of the database(visibility level of data present and set of services). A server may have implementation of certain no. of views. Databases used within the internal substation network (control network) should make use of real time memory resident feature described in the above database model. For eg.-policies, multicast address table and other records present in security hub. This will save the latencies involved in fetching the data from the disk.

5. Acknowledgements

I would like to thank Dr. Carl Gunter and Jianqing Zhang for their immense guidance and support.

6. Conclusion

In this paper, we gave a prototype design of information model of IEC 61850. We suggested use of real time main memory Database model for IEC 61850. We used TLSF memory allocator for memory allocation of records. We also gave a backup scheme for fast database crash recovery. We studied transaction processing and stated there should be assignment of priorities to transactions. We also studied security of database and emphasized on maintaining access control lists. In future, we aim to design experiments to measure latencies related to application layer.

7. References

[1]IEC 61850 Communication Networks and Systems In Substations, Technical Committee 57, International Electrotechnical Commission

[2] Secure Intelligent Electronic Devices (SIEDs).C. A. Gunter, S. T. King, J. Zhang. *PSERC 2007*

[3]Overview of IEC 61850 and Benefits, R. E. Mackiewicz. PES TD 2005/2006

[4]IEC 61850 Communication Networks and Systems In Substations: An Overview for Users. D. Baigent, M. Adamiak and R. Mackiewicz. SIPSEP 2004

[5]Main Memory Database Systems: An Overview Hector Garcia-Molina, Kenneth Salem

[6]Huang, J. and L. Gruenwald,"Crash Recovery for Real-Time Main Memory Database Systems" 1996 *ACM Symposium on Applied Computing*, pp. 145-149.

[7]DataBlitz:Main Memory DataBase System

[8]TLSF: A New Dynamic Memory Allocator for Real-Time Systems (M. Masmano, I. Ripoll, A. Crespo and J. Real). 16th *Euromicro Conference on Real-Time Systems (ECRTS 2004)*

[9]Video lectures: Advance Computer Security class By Dr. Carl Gunter Fall 2006,UIUC.

[10]Distributed systems - Concepts and Design (Fourth Edition)George Couloris Jean Dollimore,Tim Kindberg

[11] Understanding and Simulating the IEC 61850 Standard .Yingyi Liang Roy H. Campbell.IDEALS,UIUC Tech Report