Objects and Communication Services Modelling for the Automatic Speed Regulator of a typical Itaipu dam Hydraulic Turbine implementing the IEC 61850 Standard

by
David Daniel Pérez Sosa

Submitted to the Department of Electrical Engineering
in partial fulfillment of the requirements for the degree of
Electrical Engineer
at the
UNIVERSIDAD NACIONAL DEL ESTE
December 2010

Author .....
Department of Electrical Engineering
December, 2010

Certified by .....
Ladislao Aranda Arriola
Electrical Engineer, Itaipu Binacional
Thesis Supervisor

Certified by .....
Rodrigo Ramos
Electrical Engineer, Itaipu Binacional
Thesis Supervisor

Certified by .....
Juan Manuel Ramirez Duarte
Electrical Engineer, Facultad Politécnica
Thesis Supervisor

Accepted by .....
Professor Anastasio Sebastián Arce Encina
Thesis coordinator, Department of Electrical Engineering

Objects and Communication Services Modelling for the Automatic Speed Regulator of a typical Itaipu dam Hydraulic Turbine implementing the IEC 61850 Standard

by

David Daniel Pérez Sosa

Submitted to the Department of Electrical Engineering
on December, 2010, in partial fulfillment of the
requirements for the degree of
Electrical Engineer

Abstract

Forthcoming.

Thesis Supervisor: Ladislao Aranda Arriola
Title: Electrical Engineer, Itaipu Binacional

Thesis Supervisor: Rodrigo Ramos
Title: Electrical Engineer, Itaipu Binacional

Thesis Supervisor: Juan Manuel Ramirez Duarte
Title: Electrical Engineer, Facultad Politécnica

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

For example:

I would like to thank to my parents for supporting me though all of this.

Also to Leslie Lamport and Donald Knuth for LaTeX and TeX, without which I would have had much less fun formatting my thesis.

This research was supported in part by the Fundación Parque Tecnológico Itaipu supplying the needs of runaway geniuses everywhere.

Contents

List of Figures

List of Tables

Chapter 1

Work Research Overview

1.1 Introduction

For comming...

1.2 Aim of this Research

For comming...

1.3 Research Methodologies and Techniques

1.3.1 Investigación bibliográfica

Incluye la investigación y estudio de la norma IEC 61850, en especial las definiciones de modelos de información definidos en la parte IEC 61850-7-4-10 *Basic communication structure for substation and feeder equipment Compatible logical node classes and data classes - Hydro Power Plants*, los servicios de intercambio de información para diferentes funciones (por ejemplo, control, reporte, *getters* y *setters*) definidos en el apartado IEC 61850-7-2 *Basic communication structure for substation and feeder equipment Abstract communication service interface (ACSI)*, la implementación de dichos servicios de comunicación a través de protocolos especificados en la parte IEC

61850-8-1 *Specific Communication Service Mapping (SCSM) Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3* para las necesidades de la unidad generadora, y otros documentos, normas, y tecnologías relacionadas.

1.3.2 Investigación de campo

Incluye el levantamiento de informaciones del generador de Itaipú, la identificación de las funciones de automatización inherentes regulador de velocidad de la unidad generadora y otras documentaciones y estadísticas relacionadas al caso en estudio. También incluye la simulación en una red IEC 61850 utilizando las herramientas de ingeniería disponibles en el mercado para tal efecto.

1.3.3 Creación de los objetos

Diseño de los nodos lógicos normalizados en la parte IEC 61850-7-4, e IEC 61850-7-4-10 mediante UML, implementaciones en XML, Java, C++, o mediante las herramientas de ingeniería especializadas en la norma IEC 61850 disponibles en el mercado.

1.3.4 Identificación de servicios de información

Modelado completo de los servicios de comunicación necesarios para el regulador de velocidad de una unidad generadora típica de Itaipu utilizando la parte IEC 61850-7-2.

1.3.5 Propuesta de extensión/complementación de los nodos lógicos

Por ejemplo, ZAXN, insuficiente para la alimentación AC y DC de los servicios auxiliares de Itaipu.

1.3.6 Metodología de aplicación

Estructuración de una metodología de aplicación de la norma IEC 61850 en la automatización de hidroeléctricas: Elección de las herramientas de ingeniería disponibles en el mercado y que mejor se adapten a la automatización de unidades generadoras aplicando la norma IEC 61850 y la creación de procedimientos padronizados utilizando dichas herramientas, para satisfacer los requerimientos de ingeniería definidos en el apartado IEC 61850-4 subsección 3, y para identificar las funciones de automatización que son necesarias para el modelado de los nodos lógicos, pero no están contempladas en la norma IEC 61850.

1.4 Originality of this Research

For comming...

1.5 Organization of this Research Document

This text is organized as follows:

- Chapter 1: For comming...
- Chapter 2: ...

Chapter 2

Introduction to the IEC 61850

For comming...

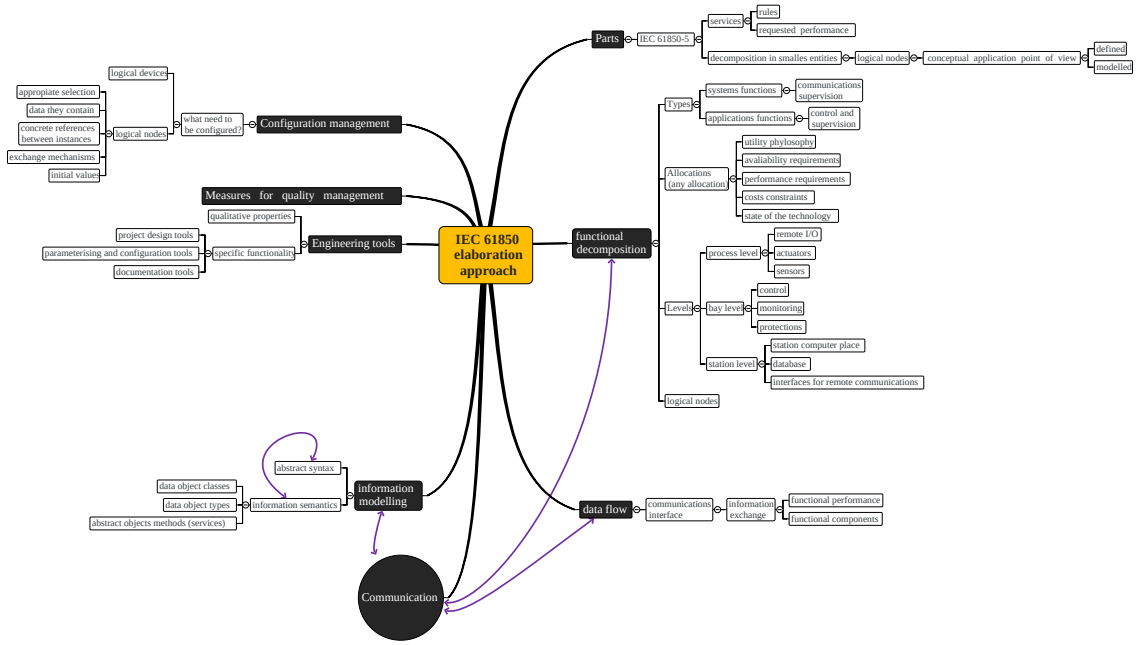

Figure 2-1: Borrador - Esquema del futuro capitulo

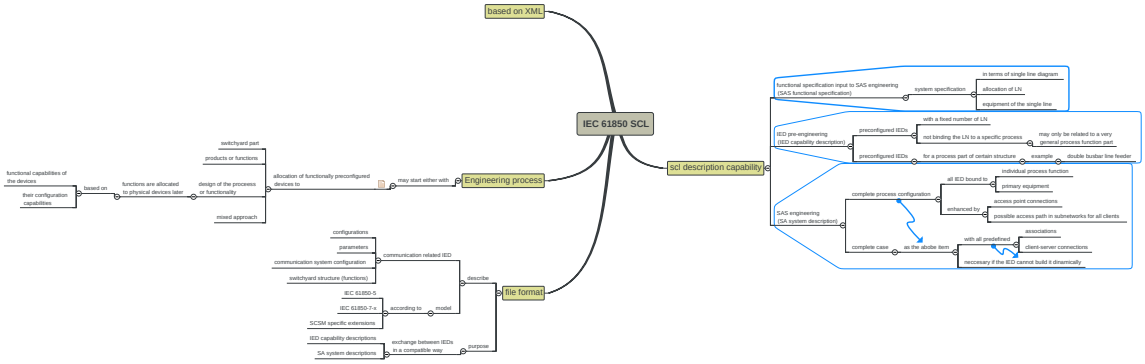

Figure 2-2: Borrador - Esquema del futuro capitulo

Chapter 3

Object-oriented system construction

3.1 Introduction

The IEC 61850 information model is classified as a object-oriented (O-O) system. Is necessary to have a clear understanding of O-O technology to dive into IEC 61850 object modelling. For this reason this chapter describes the object-oriented technology on which the IEC 61850 information model has been standarized.

The chapter does not describe all the O-O principles, just focuses on the neccesary background to understand the IEC 61850 information model. The sections which follow will define the background which the IEC 61850 object oriented information system was constructed, both formally and through examples.

A detailed description of the O-O fundamentals and reference materials are provided. To achieve a robust concepts comprehension a practical implementation with the Unified Modeling Language (UML) and Java are provided. This chapter is destined for electrical engineers and professionals of related areas which the O-O software construction is not part of the curriculum. For this reason, the practical examples uses electricity area elements.

3.2 Object Oriented systems

Many autors formulate precise definition of O-O paradigm [2] [3] [4] [5], and the definition of Lastly, Wegner [6] has been the most widely accepted [7]. Wegner defines the O-O paradigm in terms of objects, classes and inheritance.

"Objects are autonomous entities that respond to messages or operations and share a state. Classes classify objects by their common operations. Inheritance serves to classify classes by their shared behavior. Data abstraction hides the representation of data and the implementation of operations" [6]. That is:

object oriented = objects + classes + inheritance

In the next sections theses concepts are explained.

The O-O system programming metodology defines an approach to code organization for objects creation. The objects store the data and have their own behabiour with a particular information grouping by common functionality and common information structures. O-O systems benefits to bundle actions together, manage a few quantity of variables rather than multiples ones, organizing the common behavior together and structure programs in a way that matches closely the real world [8].

3.3 Classes

A class is a static template from which objects are created. Classes are used to classify objects that define common operations and a data structure, and the data types that the object can store.

Listing 3.1: Class in Java

```
public  class SERVER_v1 {


}
```

3.3.1 Attributes

Listing 3.2: Class with attributes in Java

```
public  class SERVER_v2 {

  private String[] ServiceAccessPoint;

  public String[] getServiceAccessPoint() {
    return ServiceAccessPoint;
  }
}
```

3.3.2 Methods

Methods are functions that are part of a class definition. Once an instance of the class is created, a method is bound to that instance.

Listing 3.3: Class with attributes and methods in Java

```
public  class SERVER_v3 {

  private String[] ServiceAccessPoint;

  public String[] getServiceAccessPoint() {
    return ServiceAccessPoint;
  }

  /**
   * Definido en IEC 61850 parte 7-2 (2003)
   * pgina 21, clusula 5.4, tabla 1
   * Explicado en la clusula 6.
   */
  public String GetServerDirectory(String param) {
    return null;
  }
```

```
}
```

Get and set accessor methods

Get and set accessor functions, also called getters and setters, allow you to adhere to the programming principles of information hiding and encapsulation while providing an easy-to-use programming interface for the classes that you create. Get and set functions allow you to keep your class properties private to the class, but allow users of your class to access those properties as if they were accessing a class variable instead of calling a class method. The advantage of this approach is that you can avoid having two public-facing functions for each property that allows both read and write access.

Listing 3.4: Class with attributes

```
public  class SERVER_v4 {

  private String[] ServiceAccessPoint;

  public String[] getServiceAccessPoint() {
    return ServiceAccessPoint;
  }

  public void setServiceAccessPoint(String[] serviceAccessPoint) {
    ServiceAccessPoint = serviceAccessPoint;
  }

  /**
   * Definido en IEC 61850 parte 7-2 (2003)
   * pgina 21, clusula 5.4, tabla 1
   * Explicado en la clusula 6.
   */
  public String GetServerDirectory(String param) {
    return null;
```

```
    }
}
```

Constructor methods

Constructor methods, sometimes simply called constructors, are functions that share the same name as the class in which they are defined. Any code that you include in a constructor method is executed whenever an instance of the class is created with the new keyword.

Listing 3.5: Class with attributes

```
public  class SERVER_v5 {

  public SERVER_v5() {
    // Default Constructor
  }

  public SERVER_v5(String[] serviceAccessPoint) {
    // Constructor using fields
    ServiceAccessPoint = serviceAccessPoint;
  }

  private String[] ServiceAccessPoint;

  public String[] getServiceAccessPoint() {
    return ServiceAccessPoint;
  }

  public void setServiceAccessPoint(String[] serviceAccessPoint) {
    ServiceAccessPoint = serviceAccessPoint;
  }

  /**
   * Definido en IEC 61850 parte 7-2 (2003)
```

```
     * pgina 21, clusula 5.4, tabla 1
     * Explicado en la clusula 6.
     */
    public String GetServerDirectory(String param) {
      return null;
    }
}
```

3.4 Objects

A object is a agrupation of datas and operations with autonomous memory which should inside in the behabiour of each invocation. The object datas have a state that remember the effect of the operation [6]. and unlike a class, a object just exist on a running proccess.

A object are created from a class, in other words, a object is a instance of a class.

A O-O program interaction is realized by objects invoking methods.

3.4.1 Instance

3.4.2 Reference

3.5 Diference betwen Classes and Objects

They are a clear separation between class and object, and they are (together with inheritance) the most important concepts of O-O programming languages [9].

A class is a static off-line template to generate objects. A object is a runtime data or a group of datas according to the class. The objects are independend datas with its own behavior, and they are classified by the classes which generated.

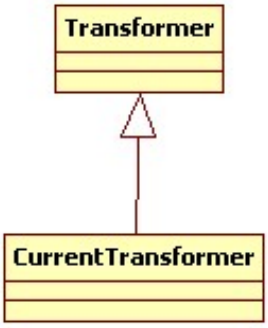

Figure 3-1: Inheritance: *Transformer* is the parent of *CurrentTransformer*

3.6 Inheritance

"The inheritance permit to a class to be used in a modified form when a sub-class is derived from it" [7]. This O-O resource can be used as a way to define new classes based in more general classes that have been defined previously, acquiring their characteristics. If a class *CurrentTransformer* directly inherits from class *Transformer* we say that *Transformer* is the parent of *CurrentTransformer* and *CurrentTransformer* is the child of *Transformer* [10]. The UML representation is given in Figure 3-1.

The inheritance is a great resource to create reusable code and common informations and methods thanks to hierarchical classes. Various techniques to write reusable code are avaliable [11] [12] [13] [14].

Listing 3.6: Generic Substation Event Class

```
public class GSE{

}
```

Listing 3.7: GOOSE Class in Java

```
public abstract class GOOSE extends GSE {

  /**
```

```
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String SendGOOSEMessage();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOReference();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOOSEElementNumber();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOCBValues();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String SetGOCBValues();

}
```

3.7 Information hidding and encapsulation

3.7.1 Abstract data types

Abstraction and information hidding form the foundation of all object-oriented development [15]. An abstraction is a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others [16]. Information hidding, as first promoted by Parnas, goes on to suggest that we should decompose systems based upon the principle of hidding design decisions about our abstractions [17] [18].

The abstraction and information hidding are very common in electrical equipments and mathematical representations of the electrical world. The models are abstracted and is possible to identify the object and operations that exist at each level of integrations Thus, when we work with phasors to represent a current, which leave just the static amplitude and phase information. The time space are hidden with the purpose to manage the information at a more hight level, thereby skipping the trigonometric calculations derived from the time dependence of the sine wave, and the information are combined just algebraically, simplifying certain kinds of complex calculations. [18]

The use of abstract data types on a object oriented system help to a more precise and at the same time simple on the specification taking adventage of the information hidding provided by abstract data types. []

Listing 3.8: Goose Abstract Class with attributes

```
public abstract class GOOSE extends GSE {

  /**
   * Definido en IEC 61850 parte 7-2 (2003)
   * pgina 21, clusula 5.4, tabla 1 <br>
   * Explicado en la clusula 15.
   */
  public abstract String SendGOOSEMessage();
```

```
    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOReference();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOOSEElementNumber();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String GetGOCBValues();

    /**
     * Definido en IEC 61850 parte 7-2 (2003)
     * pgina 21, clusula 5.4, tabla 1 <br>
     * Explicado en la clusula 15.
     */
    public abstract String SetGOCBValues();

}
```

3.7.2 Intefaces

A external interface is a group of methods declarations of a class that allows that serves to comunicate the class with other classes and thus between the designer of the class with another classes. The other clases should exist at the moment of the

interface definition or could be created in the future [8, pp. 90-105] [10].

The interface of a class serves to know how to interact with a determined behabiour of the object, their input, ouput, and optionally the error handling parameters but not knowing how the behavior is implemented.

3.7.3 Encapsulation

The encapsulation is a technique for avoid dependencies among classes and modules by defining strict interfaces, and help for software maintenance, by information hidding, understandability of code, localization of information, narrow interfaces, and information restriction [19].

Listing 3.9: Goose Abstract Class with attributes

```
public class TimeStamp{

  private int SecondSinceEpoch;

  private int FractionOfSecond;

  private java.sql.Timestamp timeStamp;

  public TimeStamp(long arg0) {
    timeStamp = new java.sql.Timestamp(arg0);
  }



  public int getSecondSinceEpoch() {
    this.SecondSinceEpoch = timeStamp.getNanos();
    return this.SecondSinceEpoch;
  }

  public void setSecondSinceEpoch(int secondSinceEpoch) {
    this.SecondSinceEpoch = secondSinceEpoch;
  }
```

```
  public int getFractionOfSecond() {
    /*
     * TODO falta implementar, ver la parte 7-2,
     * clusula 5.5.3.7.3.2
     */
    return this.FractionOfSecond;
  }

  public void setFractionOfSecond(int fractionOfSecond) {
    this.FractionOfSecond = fractionOfSecond;
  }

}
```

3.7.4 Interfaces and implementation

Interfaces are based on the distinction between a methods interface and its implementation. A methods interface includes all the information necessary to invoke that method, including the name of the method, all of its parameters, and its return type. A methods implementation includes not only the interface information, but also the executable statements that carry out the methods behavior. An interface definition contains only method interfaces, and any class that implements the interface is responsible for defining the method implementations. [8, pp. 90-105]

Listing 3.10: Class with attributes and methods in Java

```
/**
 * The SERVER class represents the external visible behaviour of
 * a device. All other ACSI models are part of the server.
 *  @category IEC 61850 parte 7-2 (2003)
 *  pgina 17, clusula 5.2 , prrafo
 *
 * @author David Prez
 *
 */
```

```
public abstract class SERVER_interface {

  private String[] ServiceAccessPoint;

  public String[] getServiceAccessPoint() {
    return ServiceAccessPoint;
  }
  public void setServiceAccessPoint(String[] serviceAccessPoint) {
    ServiceAccessPoint = serviceAccessPoint;
  }
  /**
   * Definido en IEC 61850 parte 7-2 (2003)
   * pgina 21, clusula 5.4, tabla 1
   * Explicado en la clusula 6.
   */
  public abstract String GetServerDirectory(String param);
}
```

3.8 Events

3.8.1 Event Dispatching

3.9 Exception handling

The exception is an unwanted and unexpeceted event which occours at runtime, and are caused by an unusual situation of the software that stop the normal secuence of software operations.

When a exception event are dispatched by the system, the error could be handled by a specific modelled object if the software developer anticipated it as a posibility.

The exception could be modelled with different handling stategies, [20] [21] [**?**] [22] for example, could be represented by classes specifically designed for this purpose of which concrete exceptions would be some referenceable instances that handles the

event of error.

Listing 3.11: Service Error Class in Java

```
public class ServiceError extends Exception {

  /**
   * Error de la respuesta al consumirse el
   * servicio de Logical Device
   */
  private static final long serialVersionUID = 1L;

  public ServiceError(String message) {
    super(message);
  }
}
```

3.10 Object oriented develoment approach

The object-oriented development should follow the steps proposed by Aboott [23]:

- Identify the objects and their attributes.
- Identify the operations suffered by and required of each object.
- Establish the visibility of each object in relation to other objects.
- Establish the interface of each object.
- Implement each object.

Chapter 4

Computer Networks

4.1 Introduction

The purpose of this chapter is to provide the necessary background to understand the concepts related to computer networks, focusing to explain the main concepts applied to the IEC 61850.

4.2 Transmission technologies

Types of transmission technology:

- Broadcast links.
- Point-to-point links.

Broadcast networks have a single communication channel that is shared by all the machines on the network. Short messages, called packets in certain contexts, sent by any machine are received by all the others. An address field within the packet specifies the intended recipient. Upon receiving a packet, a machine checks the address field. If the packet is intended for the receiving machine, that machine processes the packet; if the packet is intended for some other machine, it is just ignored. Some broadcast systems also support transmission to a subset of the machines, something known as multicasting [1].

In contrast, point-to-point networks, sometimes called unicasting, consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths, are possible, so finding good ones is important in point-to-point networks [1].

4.3 Adressing and routing

Each node must be able to say which of the other nodes on the network it wants to communicate with. This is done by assigning an address to each node. An address is a byte string that identies a node; that is, the network can use a nodes address to distinguish it from the other nodes connected to the network. When a source node wants the network to deliver a message to a certain destination node, it species the address of the destination node. If the sending and receiving nodes are not directly connected, then the switches and routers of the network use this address to decide how to forward the message toward the destination. The process of determining systematically how to forward messages toward the destination node based on its address is called routing [24].

Adressing types:

4.3.1 Unicast

The source node wants to send a message to a single destination node.

4.3.2 Broadcast

The source node want to *broadcast* a message to all the nodes on the network.

4.3.3 Multicast

The source node send a message to some subset of the other nodes, but not all of them.

4.4 Local Area Networks

Local area networks, generally called LANs, are privately-owned networks within a single building or campus of up to a few kilometers in size. LANs are distinguished from other kinds of networks by three characteristics:

- their size,
- their transmission technology, and
- their topology.

LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management. LANs may use a transmission technology consisting of a cable??????? to which all the machines are attached. Traditional LANs run at speeds of 10 Mbps to 100 Mbps, have low delay (microseconds or nanoseconds), and make very few errors. Newer LANs operate at up to 10 Gbps.

Various topologies are possible for broadcast LANs. Figure 4-1 shows two of them. In a bus (i.e., a linear cable) network, at any instant at most one machine is the master and is allowed to transmit. All other machines are required to refrain from sending. An arbitration mechanism is needed to resolve conflicts when two or more machines want to transmit simultaneously. The arbitration mechanism may be centralized or distributed. IEEE 802.3, popularly called Ethernet, for example, is a bus-based broadcast network with decentralized control, usually operating at 10 Mbps to 10 Gbps. Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

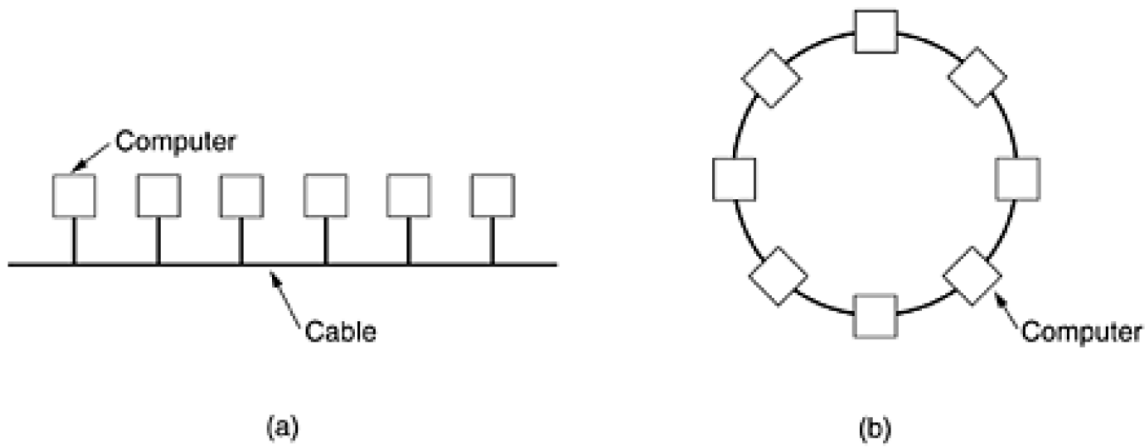

Figure 4-1: Two broadcast networks. (a) Bus. (b) Ring [1]

4.5 Common services

Is possible think of a network as providing the means for a set of application processes that are distributed over those computers to communicate. In other words, a requirement of a computer network is that the application programs that runs on the hosts connected to the network must be able to communicate in a meaningful way.

When two application programs need to communicate with each other, there are a lot of complicated things that need to happen beyond simply sending a message from one host to another. One option would be for application designers to build all that complicated functionality into each application program. However, since many applications need common services, it is much more logical to implement those common services once and then to let the application designer build the application using those services. The goal is to hide the complexity of the network from the application without overly constraining the application designer. [24, pp. 15]

4.6 The OSI Architecture

4.7 Reliability

4.7.1 Classes of failure

- When a packet is transmitted over a physical link:
  - Bit errors: 1 is turned into a 0 or vice versa.
  - Burst error: Several consecutive bits are corrupted.
- At the packet level
  - The packet contains an uncorrectable bit error and therefore has to be discarded.
  - One of the nodes that has to handle the packet is so overloaded that it has no place to store the packet, and therefore is forced to drop it.
- Node and link level
  - A physical link is cut
  - The computer it is connected to crashes. [24, pp. 18-19]

4.8 Link networks

4.8.1 Some considerations for physically conected hosts

Encoding

A link is a phisical medium carring signals and provide the foundation for transmitting all sorts of information. We say that binary data is *enconded* in the signals. [24, pp. 68]

Framming

The matter of delineating the secuence of bits transmitted over the link into complete messages that can be delivered to the end node.

Error detection

Reliability

Making a link appear reliable in spite of the fact that it corrupts frames from time to time.

Media access control problem

[24, pp. 64].

4.8.2 Links

Chapter 5

IEC 61850: Technical overview

5.1 Objects for distributed systems

The effective distribution of Logical Nodes on diferents IEDs is a reality thanks to researches about the structure of distributed systems. More than 20 years ago emerged requirements for the object paradigm to suport the design and development of distributed systems.

Theses quotes were extracted from Jazayeri 1988 research:

"An object on one node can send a (multicast) message to several other objects ..." (MCAA)

" ... The ability to group a set of objects and address them as one entity is important in many applications both from an efficiency point of view and from a program structuring point of view ..." (DO, DATA-SET, FCD, FCDA)

" ... a final difference is that our objects are active and not reactive, in the sense that they can start up spontaneosly performing operations, not necessarily only in response to method invocations. Such a facility is useful, for example, to allow objects to monitor the enviroment and change their behavior based on changes in the enviroment ..." (Some active objects are GOOSE, URCB and some passive object are)

[25].

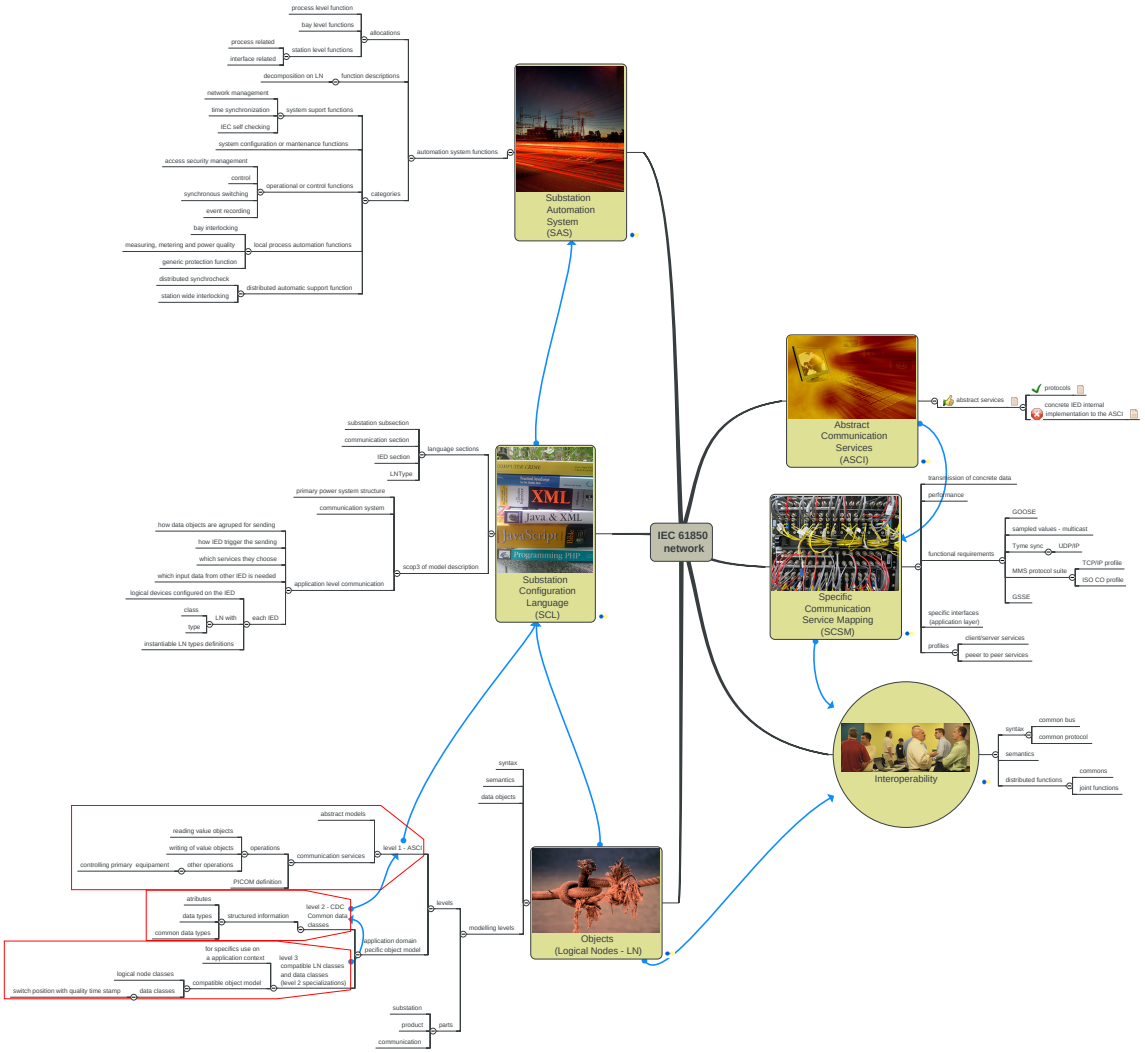

Figure 5-1: Borrador - Parte del esquema del futuro capitulo

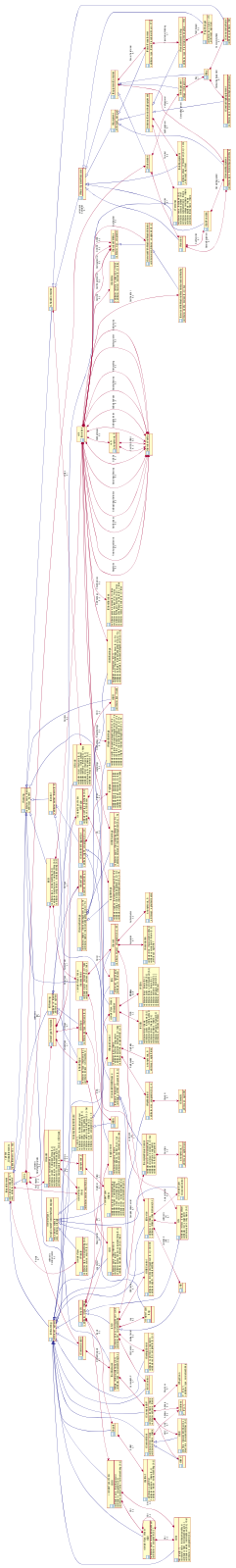

Figure 5-2: Borrador - Parte del esquema del futuro capitulo

Bibliography

[1] A. S. Tanenbaum, *Computer Networks*, 4th ed. Upper Saddle River, New Jersey 07458: Prentice Hall PTR, 2003.

[2] T. Rentsch, "Object oriented programming," *SIGPLAN Not.*, vol. 17, no. 9, pp. 51–57, 1982.

[3] P. G. A., "Elements of object-oriented programming," *Byte*, vol. 11, no. 8, pp. 139–144, 1986.

[4] K. Nygaard, "Basic concepts in object oriented programming," in *Proceedings of the 1986 SIGPLAN workshop on Object-oriented programming*. New York, NY, USA: ACM, 1986, pp. 128–132.

[5] O. L. Madsen and B. Møller-Pedersen, "What object-oriented programming may be - and what it does not have to be," in *ECOOP '88: Proceedings of the European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 1988, pp. 1–20.

[6] P. Wegner, "Dimensions of object-based language design," *SIGPLAN Not.*, vol. 22, no. 12, pp. 168–182, 1987.

[7] L. F. Capretz, "A brief history of the object-oriented approach," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, p. 6, 2003.

[8] *Programming Adobe®ActionScript®3.0*, flash_as3_programming.pdf, Adobe Systems Incorporated, 2008.

[9] O.-J. Dahl and K. Nygaard, "Simula67 common base language," *Oslo: Norwegian Computing Centre*, no. S-22, 1970.

[10] A. Snyder, "Encapsulation and inheritance in object-oriented programming languages," in *OOPLSA '86: Conference proceedings on Object-oriented programming systems, languages and applications*. New York, NY, USA: ACM, 1986, pp. 38–45.

[11] J. R. E. and F. B., "Designing reusable classes," *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22–35, 1988.

[12] M. J., "Encapsulation, reusability and extensibility in object-oriented programming languages," *Journal of Object-Oriented Programming*, vol. 1, no. 1, pp. 12–36, 1988.

[13] S. Gossain and B. Anderson, "An iterative-design model for reusable object-oriented software," in *OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 1990, pp. 12–27.

[14] L. F. Capretz and P. A. Lee, "Reusability and life cycle issues within an object-oriented methodology," in *TOOLS 8: Proceedings of the eighth international conference on Technology of object oriented languages and systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992, pp. 139–150.

[15] H. Levy, *Capability Based Computer Systems*. Bedford, MA: Digital Press, 1984.

[16] M. Shaw, "Abstraction techniques in modern programming languages," *IEEE Software*, vol. 1, pp. 10–26, 1984.

[17] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," pp. 139–150, 1979.

[18] G. Booch, "Object-oriented development," *IEEE Trans. Softw. Eng.*, vol. SE-12, Nro 2, Feb./Oct. 1986.

[19] K. Lieberherr, I. Holland, and A. Riel, "Object-oriented programming: an objective sense of style," *SIGPLAN Not.*, vol. 23, no. 11, pp. 323–334, 1988.

[20] D. Weinreb, D. Moon, and R. M. Stallman, *Lisp Machine Manual*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1983.

[21] C. Dony, "An object-oriented exception handling system for an object-oriented language," in *ECOOP '88: Proceedings of the European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 1988, pp. 146–161.

[22] G. T. Leavens, "Introduction to the literature on object-oriented design, programming, and languages," *SIGPLAN OOPS Mess.*, vol. 2, no. 4, pp. 40–53, 1991.

[23] R. J. Abbot, "Report on teaching ada," Science applications, Inc., Tech. Rep. 129, 1980.

[24] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 3rd ed. San Francisco, California: Morgan Kaufmann Publishers, 2003.

[25] M. Jazayeri, "Objects for distributed systems," in *OOPSLA/ECOOP '88: Proceedings of the 1988 ACM SIGPLAN workshop on Object-based concurrent programming*. New York, NY, USA: ACM, 1988, pp. 117–119.