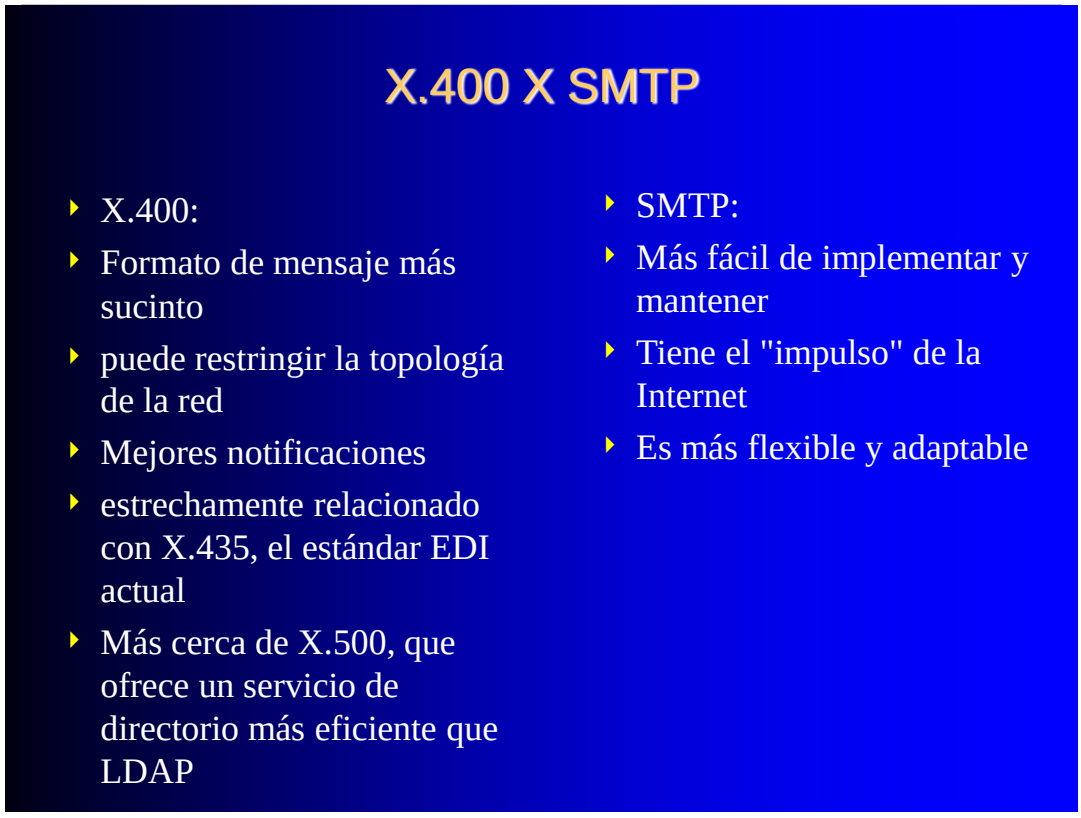
Taller AMHS

2023

Francisco Almeida – Oficial CNS SAM



Comparisons between X.400 and SMTP systems

It is worth mentioning the inevitable comparison that is made between the two deployed backbones of wider use: the SMTP of the Internet and the X.400/MOTIS. The Internet messaging service is admittedly regarded as more widely disseminated in terms of connected users. However, those users and applications that involve a "mission critical" aspect, which includes air traffic management, generally tend toward the use of MHS. John Rothon lists some technical advantages of X.400/MOTIS over SMTP [Rothon97]:

- Offers a succinct message format;

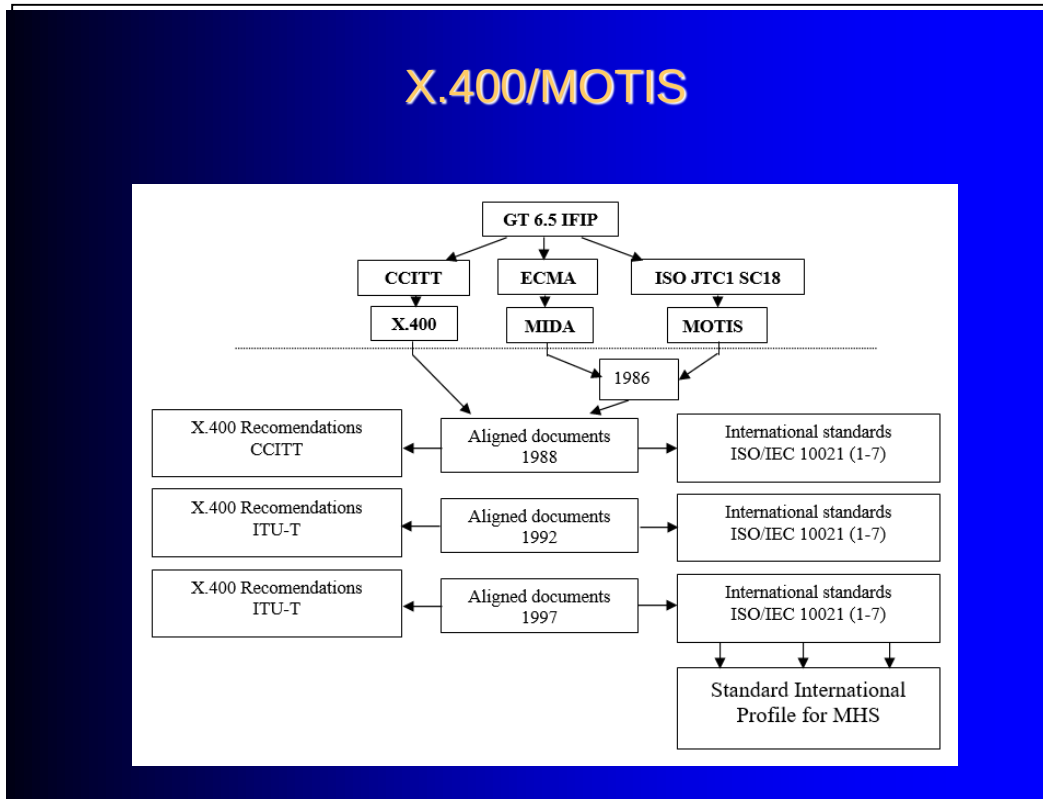
- Allows you to restrict the topology of the message backbone;

- Offers better notifications;

- It is closely affiliated with X.435 which is the current EDI standard; and,

- X.400 is more closely related to X.500 which offers a more comprehensive Directory Service than LDAP.

In a survey conducted by the Electronic Messaging Association - EMA in 1997, seventy-four percent of managers and system administrators interviewed indicated a preference for products based on X.400 recommendations in order to achieve better quality levels for their business [Rubenstein98].



The development of the X.400 Standard

In 1975, the International Federation of Information Processing – IFIP began the development of the definition of a generalized message system, through Working Group 6.5. The goal was to develop requirements for computer-based messaging systems (CBMS). Other standards organizations have adopted the idea and model of IFIP. In 1981, CCITT took over the work that culminated in the ratification, in 1984, of the X.400 series of Recommendations. In 1986, ISO decided to suspend the Draft International Standard and cooperate with CCITT to produce a joint document, ratified in 1986.

The European Computer Manufacturers Association – ECMA which had already defined the Distributed Message Exchange Application – MIDA, also joined the effort. The result of this combined effort was the joint ISO/IEC/CCITT/ECMA text [X.400], from which the ISO/IEC 10021 series of International Standards and the 1988 CCITT Blue Book of X.400 Recommendations were derived.

In 1990 and 1992, there were few changes to the existing X.400 Recommendation, but two new services were defined, one for electronic data interchange – EDI handled in the F.435 and X.435 series and one for voice messaging, with the F.440 and X.440 series.

The joint standardization modality was maintained in the 1992, 1997 and 1999 versions of the standards. Subsequently, ISO/IEC has developed the Directory Services Use Specification Standards (X.500).

MHS standards are made up of functionalities and not all functions are necessary for all users. Actually, MHS can support a variety of different layers of communication protocols. This raises the possibility that different system vendors may develop different subsets of functionality and possibly implement different protocol layers.

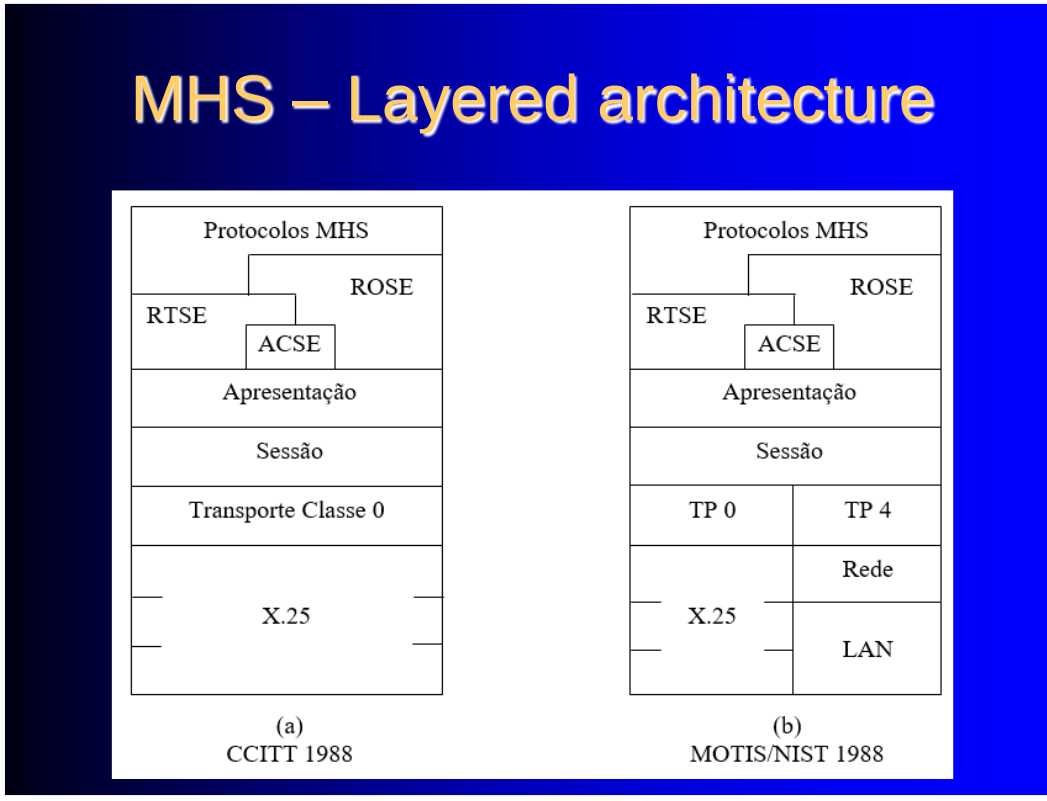
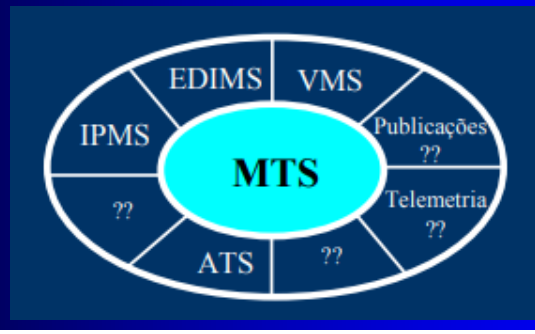


Figure (a) shows the 1988 version of the old CCITT, which uses X.25 and transport class 0 in the lower layers. Figure (b) shows the ISO 1988 MOTIS version, which can also support transport class 4 and a variety of LAN technologies in the lower layers.

MHS Services

- MHS standards include specifications for a number of services for users:
  - Message Transfer Service (MTS)
  - Interpersonal Message Service (IPM)
  - Electronic Data Interchange Messaging Service (EDI)
  - Other services



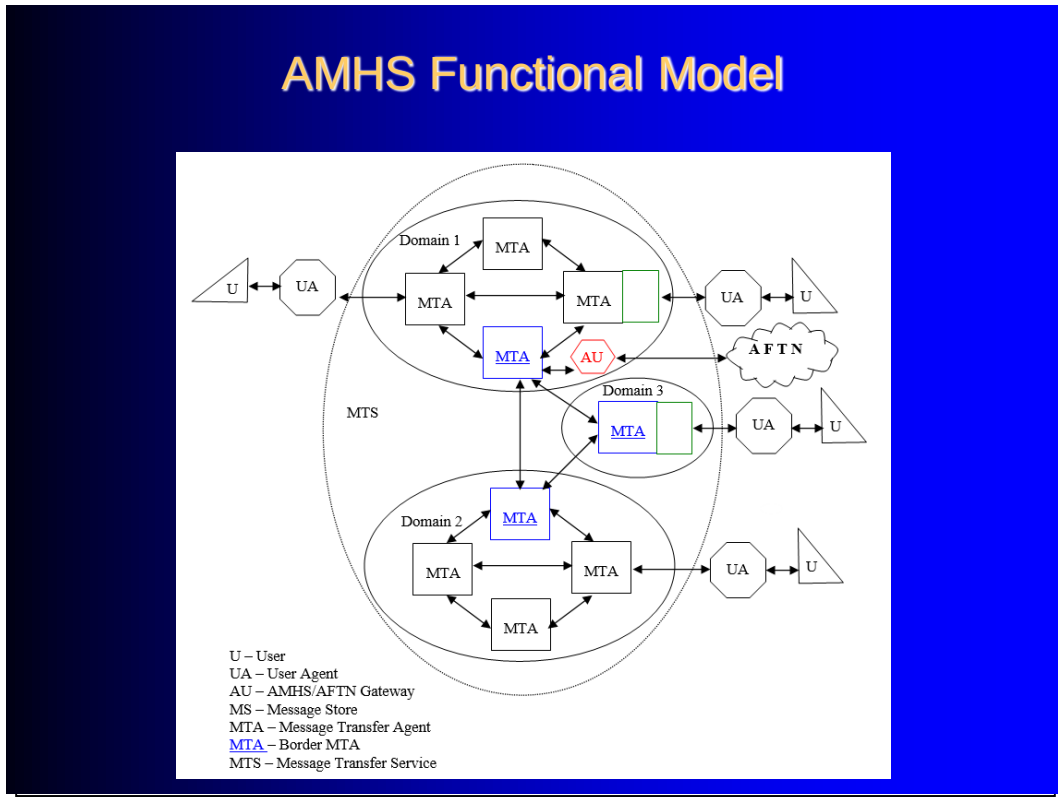
Several services have been specified in the MHS standards, which are: Message Transfer Service, Interpersonal Messaging Service (IPMS), Electronic Data Interchange Messaging Service (EDIMS), and Voice Messaging Service (VMS).

The MTS is fundamental to all other services. It is an electronic postal delivery service based on the mechanisms of storage and forward, with multiple addresses. Mainly, it interprets and acts on the information contained in the envelope (Header). It usually does not process content unless the special "content conversion" function is invoked. It is usually transparent to the content of the message and thus allows any kind of data encoding to pass. Its users can be people or machines (computers).

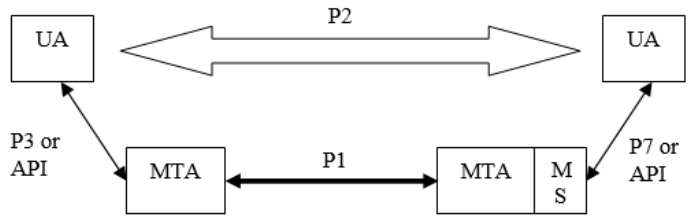
IPMS uses MTS to transfer messages between users. It offers communication between people in the style of email, personal letter and memo. Specifies a default header that allows the sender to tell the recipient what actions to take to process the message. The message may consist of one or more parts of the body. It also allows the transfer of complex information, documents and files of the most varied types, such as: spreadsheets, programs, graphics, files, etc.

Other services were defined, for example: digitized voice messaging (VMS),

Air traffic control messages (AMHS) and the store-and-foward fax messages (COMFAX).



The set of ATS message servers, user agents, and AFTN/AMHS gateways is known as the ATS Message Handling System (AMHS). The set of protocols implemented between these components is shown in the figure below. From an ATN network perspective, the three categories of systems mentioned above are ATN End Systems (ES).



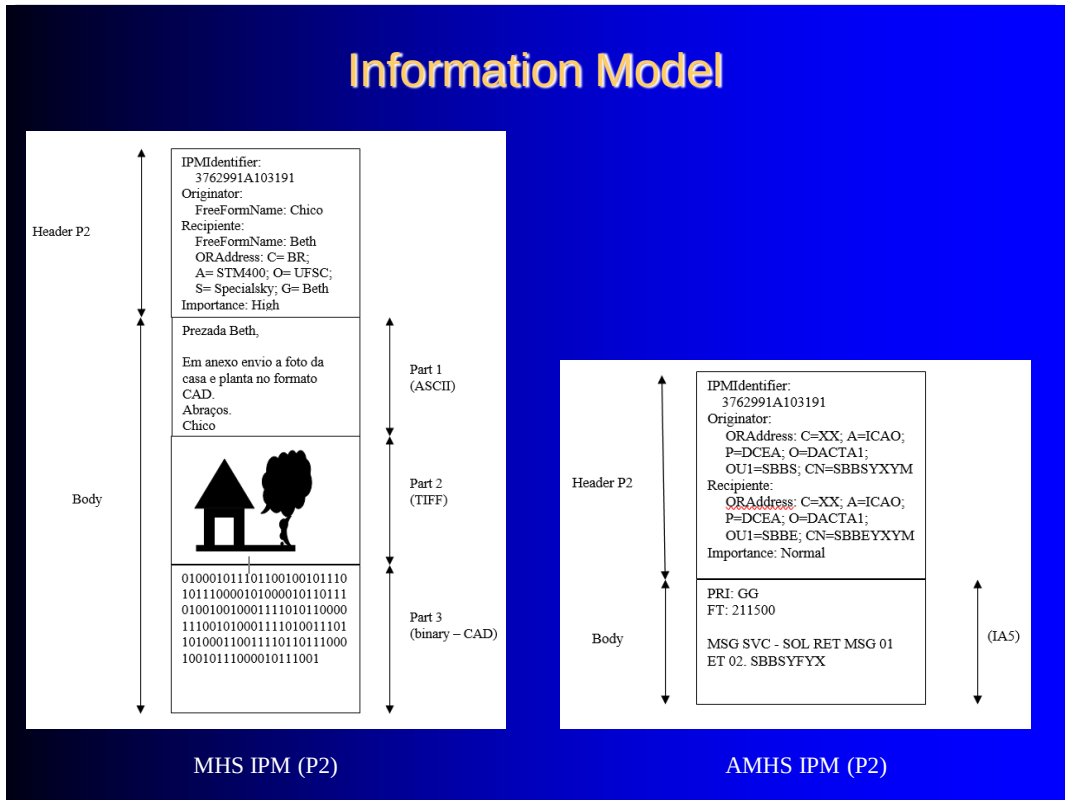
Each management domain must be interconnected with at least one other AMHS domain, which is called an adjacent domain. The concept of adjacent domains is not related to geographical considerations, but to the characteristic of direct communication between resources belonging to organizations.

Communication between two AMHS management domains is always from MTA to MTA, that is:

a) from one ATS messaging server to another ATS messaging server;

b) from an ATS messaging server to an AFTN/AMHS Gateway, and vice versa; or

c) from one AFTN/AMHS gateway to another AFTN/AMHS gateway.



In the ATS basic messaging service, the UA supports additional ATN-specific functions to meet mandatory AFTN interconnection requirements. To do this, the UA uses a structure of body parts that forwards elements that are necessary for interconnection. This structure comprises:

a) a ATS_Message_Header element, which forwards AFTN parameters that have no direct equivalents in the MHS standard; and

b) a ATS_Message_Text element, which forwards the message text itself.

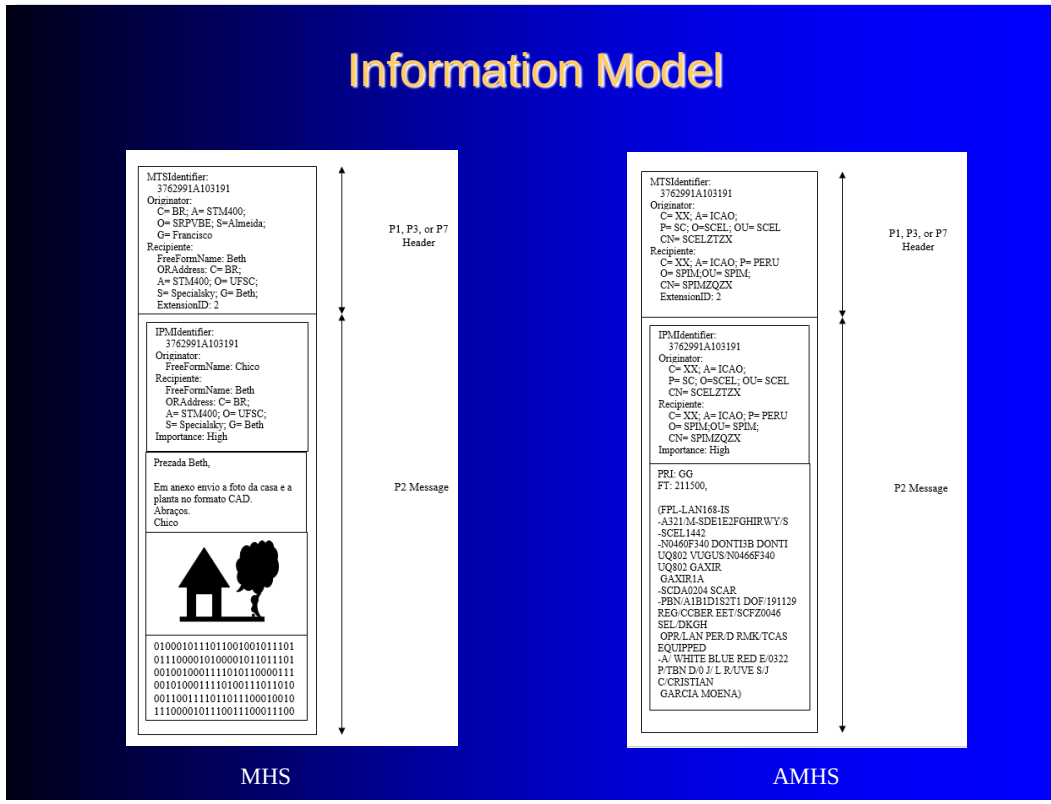
The parameters sent through the ATS_Message_Header are as follows:

a) Priority indicator, which is routed in a structure called **ATS_Message_Priority**;

b) Filling schedule - FT, which is forwarded in a structure called **ATS_Message_Filling_Time**; and

c) Optional header information: OHI, which is forwarded in a structure named **ATS_Message_Optional_Heading_Info**.

To comply with SARPs, a UA must include the ability to support these parameters. This means that the UA must be able to generate the mandatory elements, which are the **ATS_Message_Priority** and **ATS_Messagem_Filling_Time**, and optionally be able to generate the **ATS_Message_Opcional_Heading_Info**. Only messages addressed to AFTN, i.e. indirect users, require the generation of these parameters and the lack of these parameters will cause the message to be rejected by the AFTN/AMHS gateway.



The ATS Message Handling System (AMHS) is a subset of MHS.

In the Basic Service, there is communication with indirect users (AFTN users), being necessary the inclusion of the elements of the **ATS_Message_Header** (**PRI:**, **FT:** and **OHI**) in the body of the IPM.

AMHS uses particular address schemes.

Currently, two schemes are used CAAS and XF.

COMMON AMHS ADDRESS SCHEME (CAAS)

An address in the CAAS format is composed of the following attributes:

C=XX (Country=XX that was allocated for use by ICAO Member States);

A=ICAO (ICAO Administrative Domain);

P=variable (Private domain of a State or Organization);

O=variable (Organization in a State – usually the MTA location);

OU=variable (Organizational unit – usually an ATS center); and

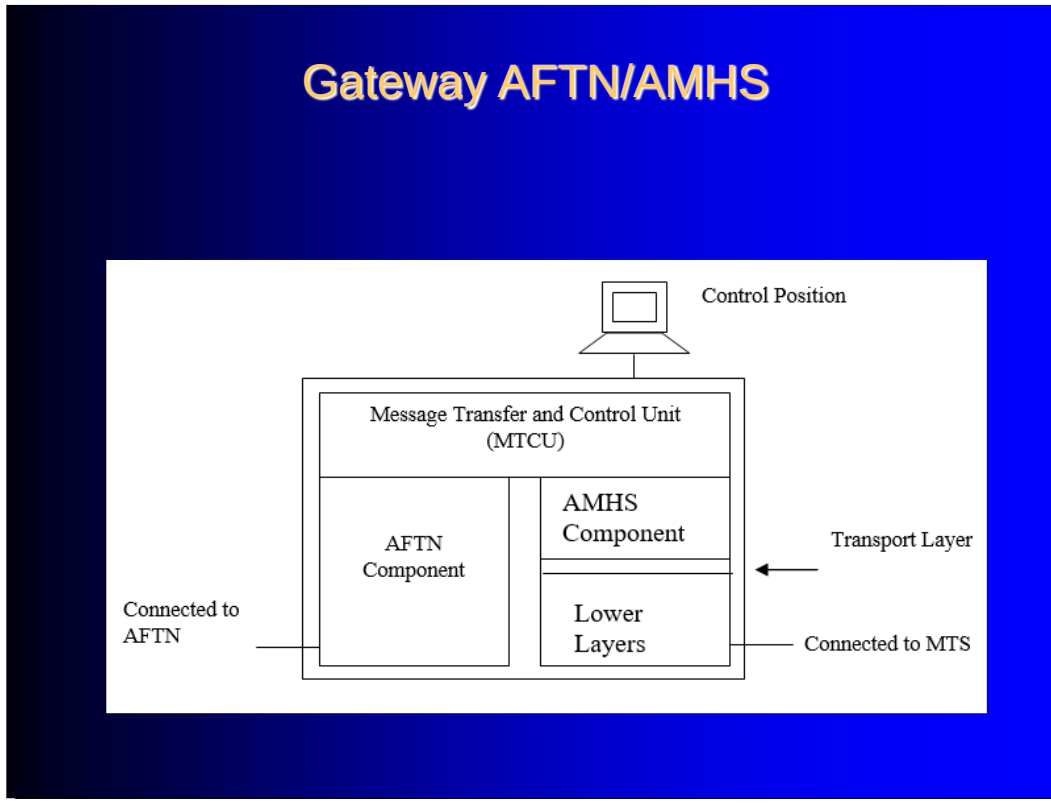
CN=variable (Common name – usually the old AFTN address).

Example: /C=XX/A=ICAO/P=SPAIN/O=LECS/OU=LEMG/CN=LEMGZPZX/

XF SCHEME

An XF address has the attribute O=AFTN and the OU attribute is the old AFTN address of the ATS center (the CN attribute is not used).

Ejemplo: /C=XX/A=ICAO/P=K/O=AFTN/OU=KATLZPZX/



Three main components define the gateway: the AFTN component, the ATN component, and the message transfer and control unit (UCTM).

AFTN COMPONENT

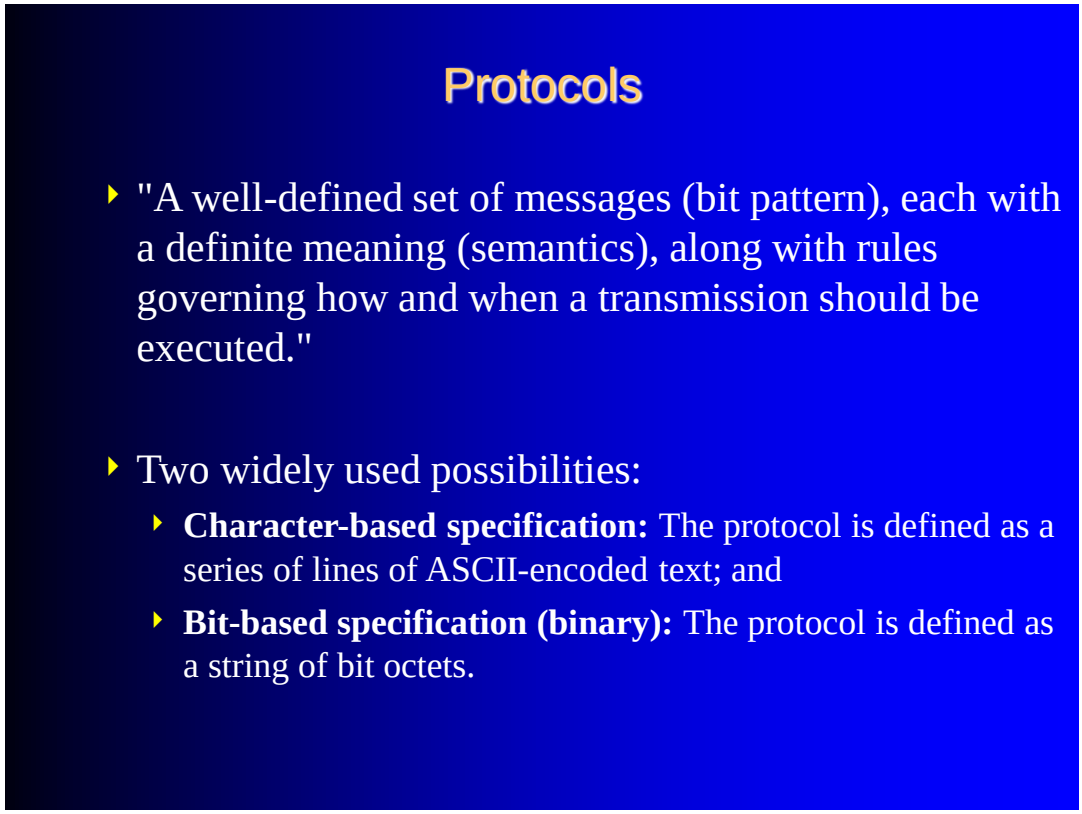
The AFTN component establishes a full connection from the gateway to an AFTN center, with the ability to send and receive messages. The gateway must operate with a full set of functions, appearing to the adjacent center as an AFTN station. An AFTN address must be assigned to the AFTN component.

AMHS COMPONENT

The ATN component allows the gateway to function as an ATN end system. Equivalent to the ATS messaging server, the ATN component incorporates an MTA. This MTA must implement the distribution list functional group according to the specifications of the ATS messaging server. Optionally, the management domain can implement other optional functionality groups within the ATN component of the gateway.

MESSAGE TRANSFER AND CONTROL UNIT

In the AFTN/AMHS gateway, the message transfer and control unit (MTCU) provides application-level functions related to the MHS Access Unit (AU), which are not part of the AFTN component or the ATN component. These functions connect and integrate the other two components and are essential for the operation of the gateway.



A protocol can be defined as:

"A well-defined set of messages (bit pattern), each with a definite meaning (semantics), along with rules governing how and when a transmission should be executed."

A protocol rarely exists alone. It is usually part of a protocol stack, in which several different specifications work together to determine the complete message sent by the sender, with some parts of this message intended for action by intermediate nodes (intermediate systems) and other parts directed to the remote end system.

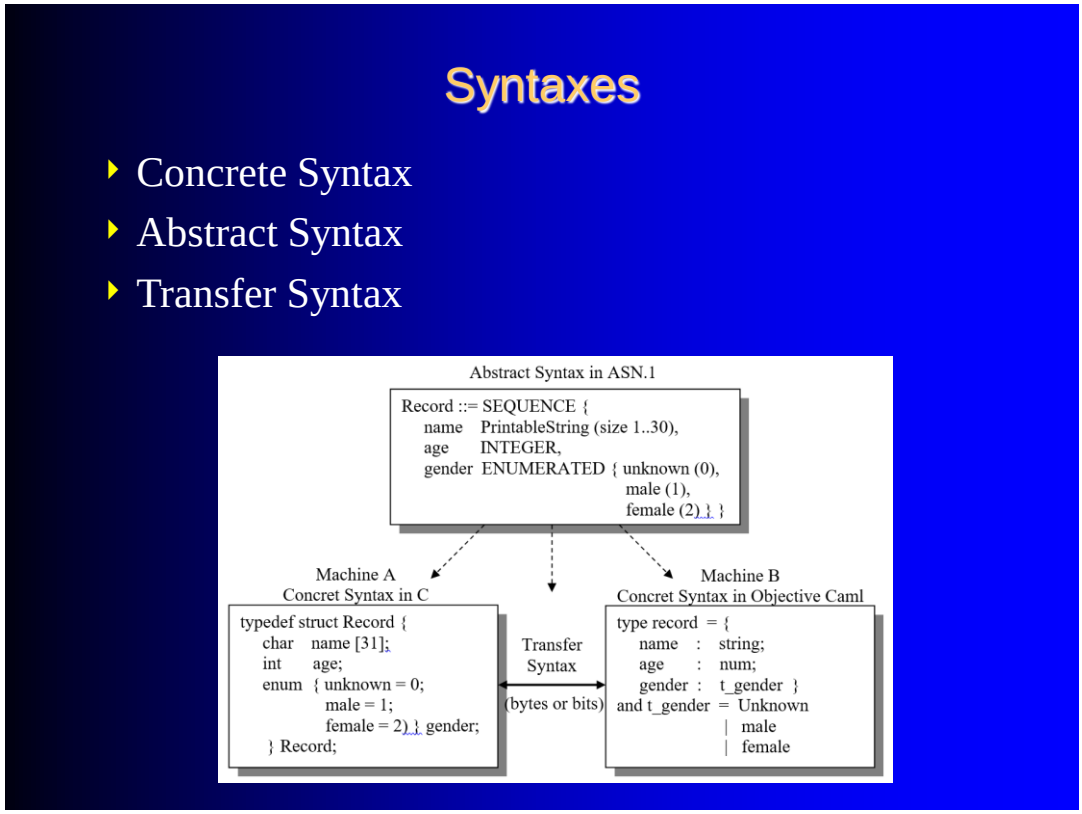
PROTOCOLS SPECIFICATION

Protocols can be specified in several ways. Two possibilities that are widely used and that maintain fundamental distinctions between them are:

Character-based specification: the protocol is defined as a series of lines of ASCII-encoded text; and

Bit-based specification (binary): the protocol is defined as a string of bit octets.

The second is known as the **abstract syntax approach**. This is the approach used by **ASN.1**, which has the advantage of allowing the developer to produce specifications without worrying about coding details; and also allows the use of tools for easy implementation of specified protocols. In addition, because it is coding-independent, it facilitates migration to more advanced encodings when they are developed.



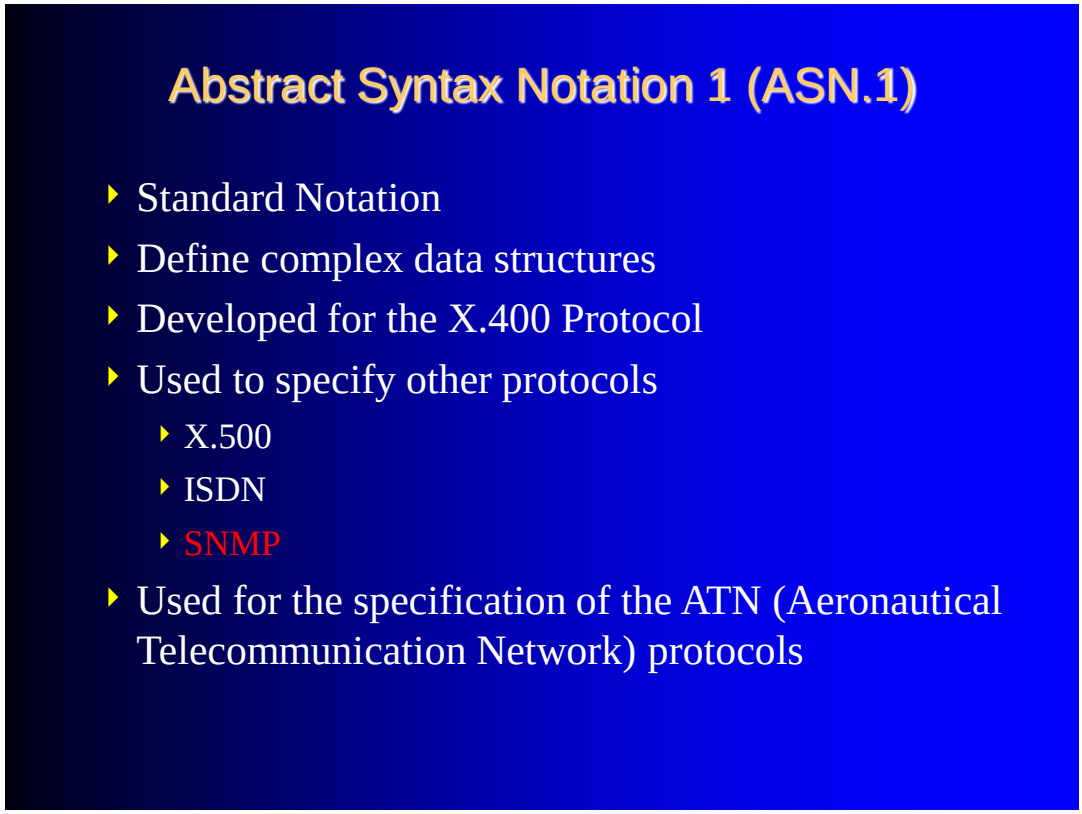
Concrete syntax: concrete syntax is the representation of the data structures that will be transferred in a given programming language. It is a syntax because it respects lexical and grammatical rules (C, for example); It is called concrete because it is actually handled by the application (implemented in the same language) and is compatible with the constraints of the machine (hardware).

Abstract syntax: to be independent of the various concrete syntaxes (C, C++, Pascal, Fortran, etc.), the data structures to be transmitted must be described without connection to the programming language used. This description must also respect the lexical and grammatical rules of a "certain language", but must remain independent of programming languages and must never be implemented directly on the machine. For these reasons, such a description is called abstract syntax, and ASN.1 is the "language" by which this abstract syntax is denoted.

Transfer Syntax: data is received as a string of bytes or bits, which must be compatible with a syntax called transfer syntax, so that these strings can be recognized correctly by the machine.

Of course, this transfer syntax depends entirely on the abstract syntax, as it assembles how the data should be transmitted, according to the abstract syntax. In fact, the transfer syntax structures and sorts the bytes that will be sent to another machine. But unlike abstract syntax it is a physical quantity, and because of this, you need to take into account the order of bytes, weight of bits, etc.

Different transfer syntaxes can be associated with a single abstract syntax. This is particularly interesting when the data flow (throughput) increases and more complex coding becomes necessary. In such a case, it is possible to change the transfer syntax without changing the abstract syntax.



Abstract syntax notation 1 (ASN.1) is a standard notation for representing data structures in an implementation-independent manner. It was originally developed by CCITT to define the complex data structures employed in the X.400 protocols, and was later adopted by ISO standards writers to define the protocol data units used in OSI application protocols. Within the ATN, ASN. 1 has been used for the formal definition of application messages and for the definition of the higher-layer protocol data units used by the efficiency improvements of the upper layer. The ASN.1 standards are ISO/IEC 8824-1 and ITU-T Rec. X.208.

Primitives Types

ASN.1 relies on several primitive data types and mechanisms to construct structured types from these primitives to suit application requirements. ASN.1 also allows structured types to be defined recursively to construct additional and increasingly complex structured types. Types are named and defined by type assignment statements, and value assignment statements allow you to assign values to a specific instance of any primitive type or to a structured type of arbitrary complexity.

Defined Types

For any of the ASN.1 primitive types, simple types can be defined using type-mapping statements to effectively rename a primitive type and, optionally, define the domain of values that the type can take.

Structured Types

The ASN.1 mechanisms for defining structured types are lists, repetitions, and alternatives.

The SEQUENCE and SET keywords are used to define structured types that consist of a set of other types, either simple or self-structured. The difference between the two is that in the SET type, there is no significance to the order in which the subtypes are listed, whereas in the SEQUENCE type, the values will always be assigned in the order in which the subtypes are defined.

ASN.1 Type Classes

- Universal
- Wide application
- Context-specific
- Private

Each type is distinguished by a label, which specifies the class of the type and identifies the type particularly. For example, UNIVERSAL 4 refers to OctetString, which is of the UNIVERSAL class and has ID 4 within that class.

ASN.1 as described in Recommendation X.409 has four classes of types:

Universal: types of general utility, regardless of application and construction mechanisms; These are defined in the standard and listed in the table below.

TAG	TYPE	TAG	TYPE
UNIVERSAL 1	Boolean	UNIVERSAL 18	NumericString (Character String)
UNIVERSAL 2	Integer	UNIVERSAL 19	Printable String (Character String)
UNIVERSAL 3	BitString	UNIVERSAL 20	TeletexString (Character String)
UNIVERSAL 4	OctectString	UNIVERSAL 21	VideotexString (Character String)
UNIVERSAL 5	Null	UNIVERSAL 22	IA5String (Character String)
UNIVERSAL 6	Object Identifier	UNIVERSAL 23	UTCTime
UNIVERSAL 7	Object Descriptor	UNIVERSAL 24	Generalized Time
UNIVERSAL 8	External	UNIVERSAL 25	Graphic String (Character String)
UNIVERSAL 9-15	Reserved	UNIVERSAL 26	Visible String (Character String)
UNIVERSAL 16	Sequence and Sequence-of	UNIVERSAL 27	General String (Character String)
UNIVERSAL 17	Set and Set-of	UNIVERSAL 28	Reserved

Wide application: relevant to a specific application; These are defined in other standards.

Context specific: also relevant to a particular application, but applicable to a limited context.

Private: User-defined types that are not covered by any standards.

Syntax is used both to define data types and to specify data values. The best way to describe the syntax is with an example.

Let's take the data from a Personnel Record of a fictitious company. Described without ASN.1 formality on the slide above.

Using ASN.1 to describe the data structure, you would have:

PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {

Name,

title [0] IA5String,

EmployeeNumber,

dateOfHire [1] Date,

nameOfSpouse [2] Name,

[3] IMPLICIT SEQUENCE OF ChildInformation DEFAULT {}}

ChildInformation ::= SET {

Name,

dateOfBirth [0] Date}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {

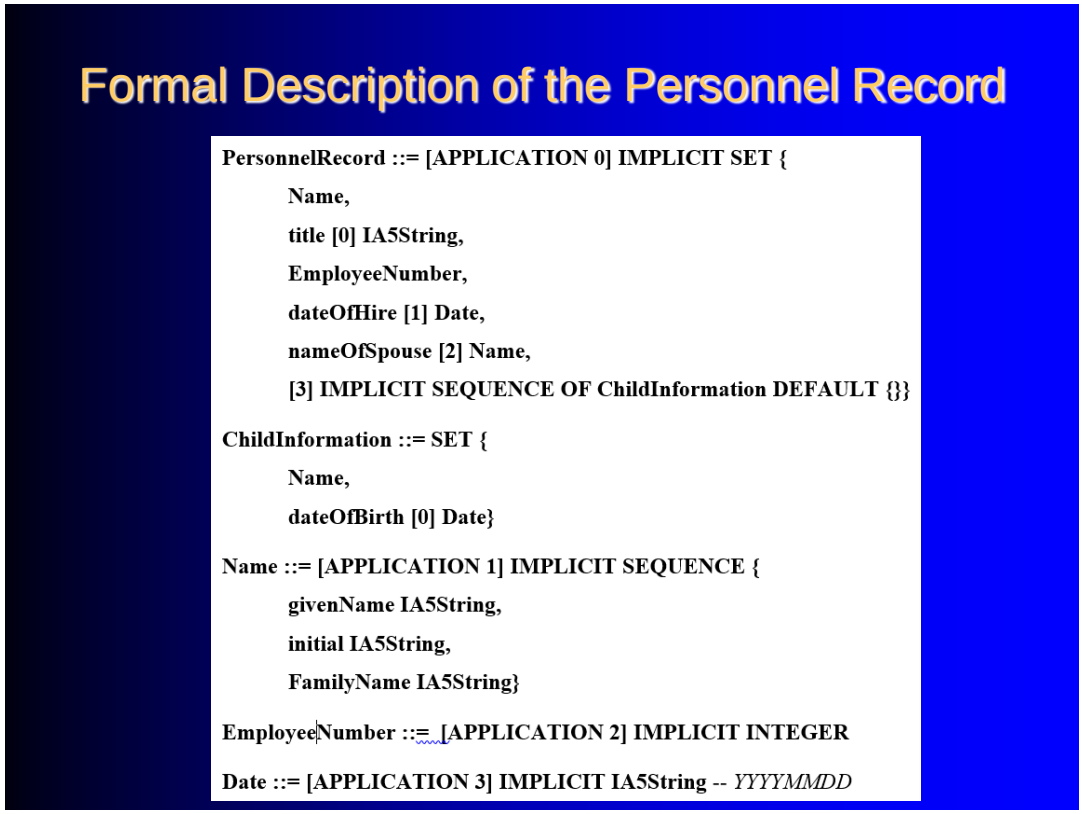
givenName IA5String,

initial IA5String,

FamilyName IA5String}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT IA5String -- YYYYMMDD



The **EmployeeNumber ::= {APPLICATION 2] IMPLICIT INTEGER** setting uses the built-in INTEGER type, but the user has chosen to assign a new label to the type. The use of the term [APPLICATION 2] provides a label (class and ID) for this new type. The implicit designation is related to the representation of values in the transfer syntax. With that term present, values of this type will be encoded only with the APPLICATION 2 tag. If the designation was not present, the values would be encoded by the APPLICATION and UNIVERSAL labels. Using the implicit option results in a more compact representation.

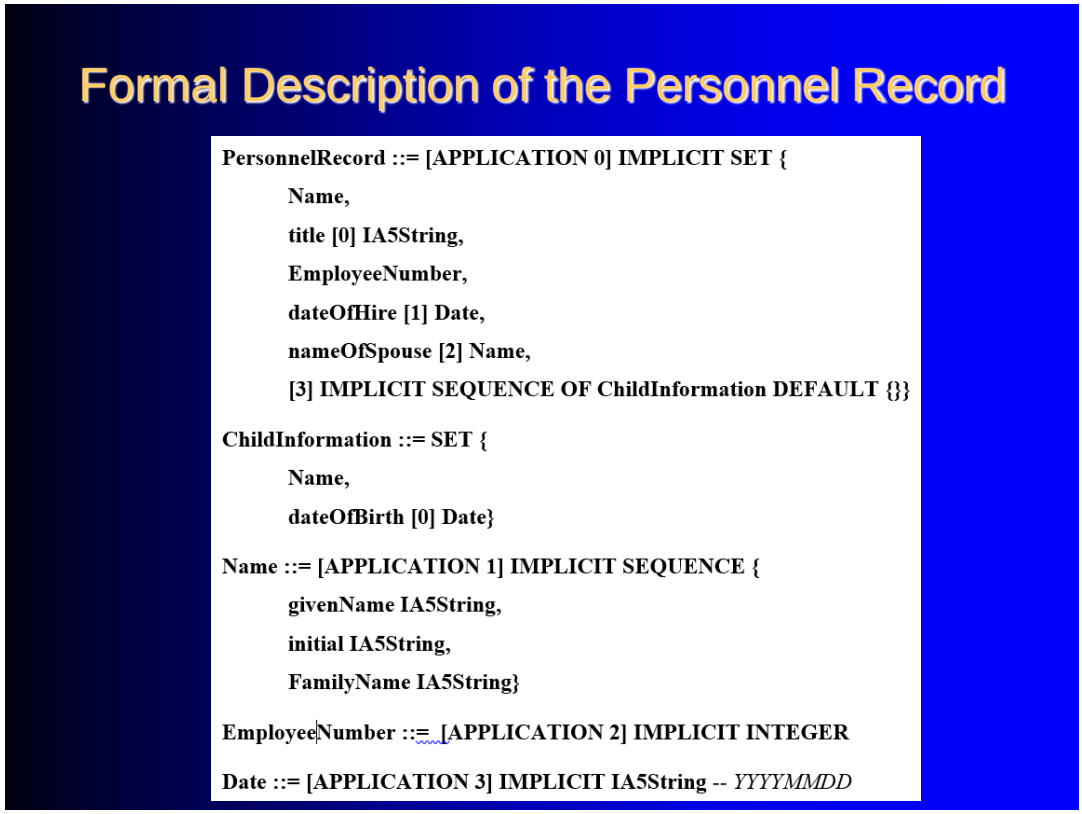
The definition of the **Date** type is similar to that of the **EmployeeNumber** type. In this case, the type is a string consisting of the character set known as IA5. The double hyphen indicates that the rest of the line is commentary; the format of values of type Date will not be checked beyond determining that the value is an IA5 string.

The other types are more complex. Consider setting the **Name** type. The basic type of **Name** is a SEQUENCE, which defines an ordered series of other types. A sequence is analogous to the file structure found in many programming languages, such as COBOL, for example. The beginning and end of the sequence definition are marked by braces. A sequence consists of a series of elements, which specify a type. Three forms of SEQUENCE are allowed:

- a variable number of elements, all of one type (the designation SEQUENCE OF is used);

- a fixed number of elements, possibly of more than one type; and

- a fixed number of elements, some of which are optional; where all elements, including optional ones, must be of different types.

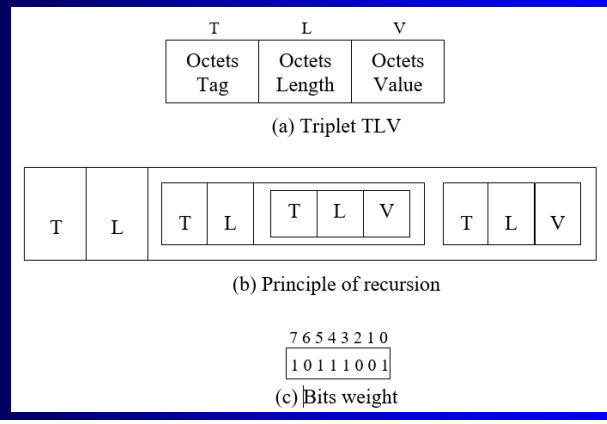


Now consider the definition of **ChildInformation**, which is a **SET** type. A SET is similar to a SEQUENCE, except that the order of the elements is not significant. Elements can be arranged in any order when encoded in a specific representation. As in Sequence, all three ways mentioned above are allowed. In the example, Set contains two elements. The first is the Name data type, which is defined in the sequence already discussed. This is an example of the recursive ability to define other typed types. Note that the first element is not given a name, but the second element is given the name **dateOfBirth**. The second element is the **Date** data type defined elsewhere. This type is used in two different contexts, here in the **PersonnelRecord** definition. In each context, the data type is given a name and a context-specific tag [0] and [1], respectively.

Finally, the overall structure, **PersonnelRecord** is defined as a **SET** with five elements. Associated with the last element is a default value of a null string, which will be used if no value is provided.

Basic Encoding Rules

The Basic Encoding Rules (BER) transfer syntax is always in the format of a TLV triplet (Type, Lenght and Value) also expressed as <Tag, Length,Value>. All fields T, L, and V are series of octets. The field V can have in itself another triplet T, L and V, if it is a constructed type, see figure (b).

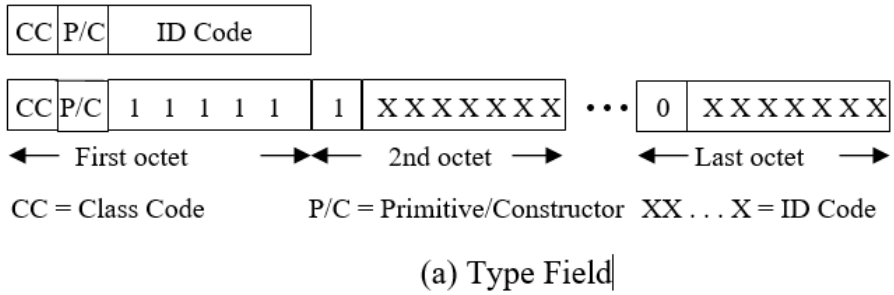


The Type field encodes the label (Tag – class and number) of the data type value. The first two bits indicate which of the four classes (universal, wide application, context-specific, or private) the types belong to. The next bit indicates whether the shape of the type is primitive or constructor.

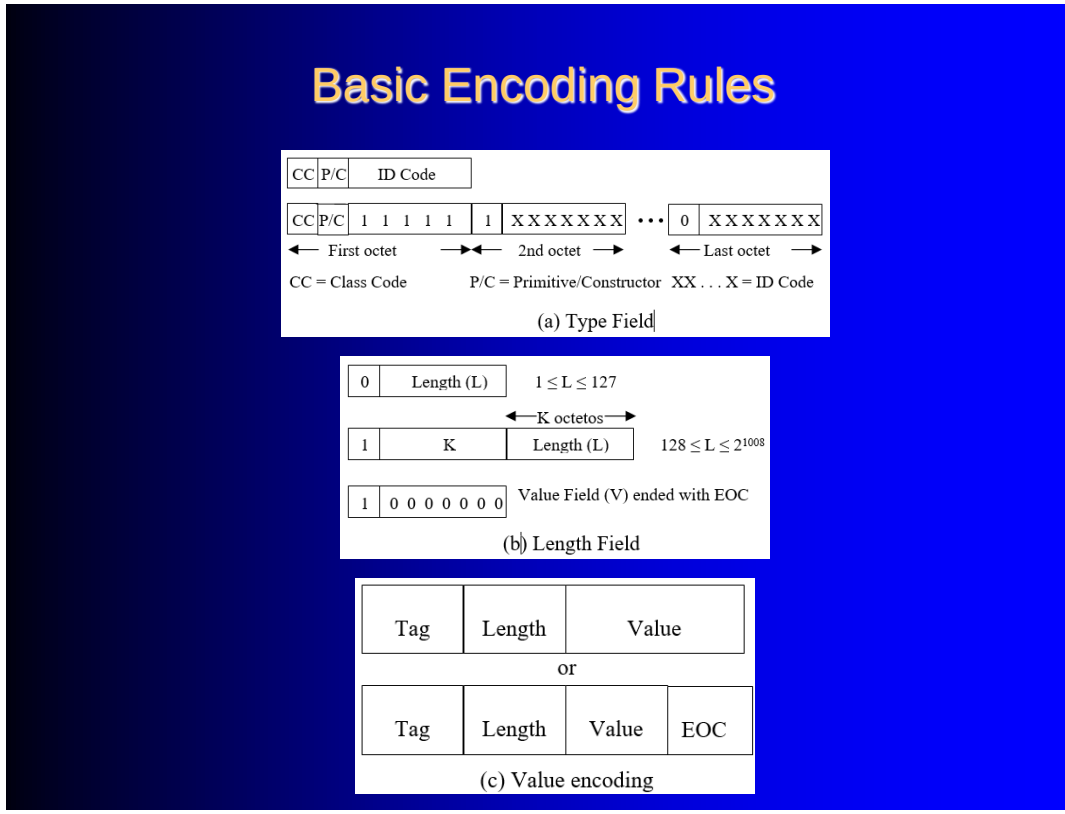
- **Primitive (basic):** contents of the Value field directly represent the value; and

- **Constructor:** contents of the Value field are the complete encoding of one or more data values. It is used for types such as Sequence and Set.

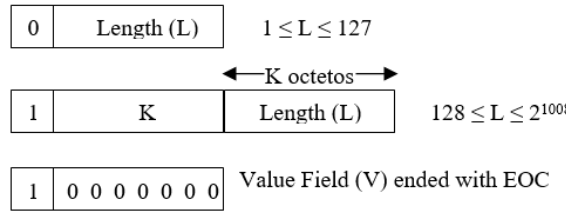
The remaining five bits of the Type field can encode a numeric ID code that distinguishes one data type from another type within the designated class. For tags whose number is greater than 31, these five bits contain the binary value 11111 and the ID is contained in the last seven bits of one or more additional octets. the first bit of each additional octet indicates whether it is the last additional octet in the Type field.



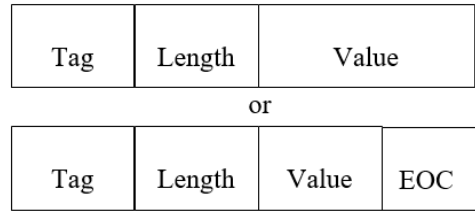
(a) Type Field



The Length field specifies the octet size of the Value field. For L less than 128, the Length field consists of a single octet starting with zero. If the length of the Value field is not immediately known, but is greater than 127, the first octet of the Length field contains a seven-bit code that specifies the size of the L field. If the size of the Value field is not immediately known, the L field takes a binary value of 10000000 and the Value field ends with an End of Content (EOC) marker.



(b) Length Field



(c) Value encoding