ICD Management (ICDM) tool for embedded systems on aircrafts

Miguel Ángel Mozas Pajares[1], Carlos Murciano Díaz[1], Ismael Lafoz Pastor[1], Carlos Fernández de la Hoz[1]

1: AIRBUS MILITARY, Paseo John Lennon, s/n 28906 Getafe (Spain)

Abstract: In the scope of on-board systems development and integration for aircrafts, ICDs (Interface Control Document/Description) are fundamental sources of information at same level that the functional requirements.

ICD description must include feasible (bus capacity not exceeded, processing load under specific limits) and accurate (data precision, data retransmission rate, data consistency) inter-connectivity mechanisms, to support the application requirements needs, as well as the information must be easy to update and to maintain.

On the other hand, DOORS (IBM) is a tool specialized on requirements management. It faces with aspects as traceability, version control, security (access control), changes control, data persistency and accessibility. DOORS is widely used and it is recognized as a reference tool in the scope of aeronautic developments.

Nevertheless, ICD information usually is not managed with DOORS. The reason can be found in the specificities of the nature of ICD information, which suggests other approaches more suitable than DOORS.

The alternative approaches are very diverse but most of them focus on handling information typified according to specific meta-models for ICD: databases, customized spreadsheets, and tabular descriptions. But whatever is the solution, it must address also, by its own means, the features of ICD derived of its “requirement nature”.

This paper presents a solution based on the integration of DOORS with a user front-end developed with Model Driven Architecture (MDA) and Open Source technologies. It is intended to provide means to manage ICD data in a robust way (guaranteeing the correctness -syntactic, completeness, non ambiguity- of data, changes, versions, dependencies, …) and additionally, to provide mechanisms to focus the effort on the data transformation rather than on the manual elaboration of derived artefacts.

Keywords: ICD managenent, tool integration, requirement management.

1. Introduction

ICD Management System (ICDMS) is an infrastructure aimed to manage the edition, the publication and the operation of ICD data in an error free, controlled and efficient manner.
The main features are:

- ICD edition
- ICD data verification
- ICD data analysis
- ICD data transformation
- ICD documentation
- Master repository
- Access control
- Configuration and change control
- Traceability
- Reuse management

1.1 ICD into the Aircraft Development Life-cycle

ICD is a key piece of system definition and system development. Its relevance comes from the role it plays in the different phases of the development process: ICD data is a requirement for the Equipment (Software and Hardware) under development. Additionally, it can be used as an input for definition of Integration test, for definition of Equipment/Software test, and for testing (Test Bench configuration, including bus analyzers and Flight Test instrumentation).

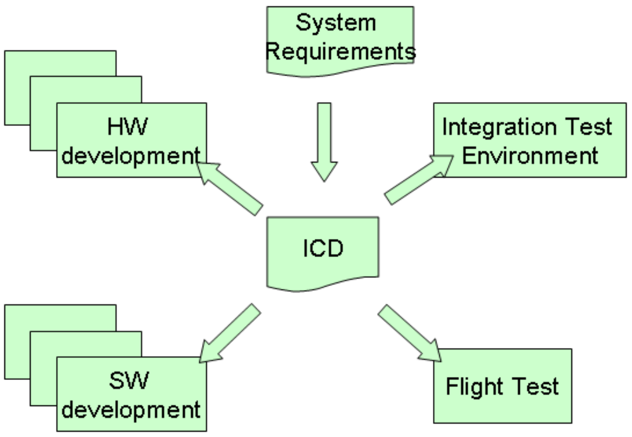

Figure 1: Impact of ICD in the development life-cycle

Because of its position in the development chain, errors or misunderstandings on ICD impact widely along the development process. The early detection of incorrectness and incompletness becomes an essential need to guarantee the efficiency of the development activities.

1.2 Nature of ICD data

ICD information is strongly **typified**. Practically any ICD description fits with one among a limited group of description pattern.

For complex systems, the number of information elements which are associated with ICD is very high (hundred of thousands of items).

Inside an organization involved in aircraft or systems development, it is quite usual to face with the need of **reusing** ICD information from one development to another, either exactly the same information or a **variant** of it.

As it has been said, ICD is a requirements source for Subsystems and Equipments. In the other side, ICD describes design decisions intended to satisfy upper level requirements. Therefore, ICD is deeply embedded in the **requirements** chain.

Sometimes, ICD data have a high degree of **volatility** along the life cycle of the system development.

ICD volatility acts like a multiplicative factor over the normal error tendency.

1.3 ICD data operation

The artefacts derived from the ICD information are strongly typified as well. The elements carrying out the transmission are typically in charge of the transmission and reception themselves, of the packaging and un-packaging of the information, and of the transformation of raw data to engineering units, as handled by the computer (transference syntax <-> local syntax). The construction of the test environment and the definition of test cases use exhaustively and systematically the ICD information.

1.4 Challenges and opportunities

Taken into account the characteristics mentioned previously, it is possible to identify important opportunities to optimise the efficiency in the management and the usage of ICD information.

The early detection of errors in the phase of ICD definition avoids the propagation of those throughout the development process where, on the other hand, it is more difficult the detection and diagnosis of the problems.

The origin of the problems can be of different nature: errors in the definition, inconsistency, incompletness, errors in the version control, errors in the control of changes or errors in the interpretation of the information, derived from the ambiguity in the expression of ICD definitions.

Some of these causes can be controlled by means of classic procedures for change control, version control and traceability (requirement management). Nevertheless, the ability to avoid other problems depends on the capability to analyse the correctness of the ICD definition.

Thus, the definition of a formal language (ICD meta-model), specialised for ICD description, appears like a fundamental tool to allow the definition precise and no ambiguous, of ICD. It is useful to facilitate the early verification of ICD, but also to analyse the implementation feasibility (load analysis), and to generate, systematically and automatically, the artefacts produced from the ICD.

Taking into account the massive volume of information being involved, such automatic generation can suppose very significant savings in the development efforts.

Moreover, the effective management of the ICD reusability allows saving important efforts on definition, development and verification.

1.5 Development environment/process extension

The purpose of the proposed approach is to achieve a powerful environment to manage ICD, by means of the integration of well-known, available and established tools, technologies and processes. The idea is to take advantage of the knowledge in on-going practices and tools, and to extend it in a smooth and undisruptive way.

In any case, we would like to emphasize the idea that having the appropriate language (formal), and the appropriated means, the artisanal way of developing ICD dependent products, can be substituted by a formalized practice that allows to improve drastically the efficiency, by providing powerful helps for correct ICD definition, and for the automatic creation of derived products (transformations). In fact, we are pointing the opportunity of introducing MDA on these scenarios.

1.6 ICDMS extension

Obviously, the current solution does not support all the possible ICD definition casuistry. Taken it into account, the development has been undertaken by applying an MDA approach, in order to be able to extend the ICDMS capabilities by working, basically, in the ICD meta-model extension.

2. Technology

The proposed solution is based on the integration of the following technologies and tools:

2.1 DOORS

DOORS 8.2 is used as Master repository with Requirements Management capabilities

- Access control,
- user management,

- change/version Management,
- traceability …

Why DOORS? We will not try to justify the use of DOORS as opposed to other alternatives in basis of its intrinsic features, but in fact, it is probably the most used tool for requirements management, in the scope of aeronautic systems development.

The interconnection between DOORS and ICD-UI has been defined to keep them as uncoupled as possible, in order to allow the substitution of DOORS by another alternative product.

2.2 Eclipse

Eclipse is a multi-language software development environment comprising an IDE and an extensible plug-in system.

2.3 Eclipse Modelling Framework (EMF)

The **EMF** project is a modelling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor

EMF framework includes a meta-model (**Ecore**) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization and a very efficient reflective API for manipulating EMF objects generically.

The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility supports regeneration of code while preserving user modifications.

2.4 Object Constraint Language (OCL)

OCL is a formal and declarative language that allows defining constraints any meta-model that cannot otherwise be expressed by diagrammatic notation. It is used to complement Ecore meta-model definition.

It also allows defining model analysis algorithms, and elaborating object query (filter) expressions. These features are used for implementing the ICDMS “analysis” and “search” capabilities in a declarative –no programmatic- way, closely connected to ICD meta-model definition.

In fact, when used for defining model constraints and model analysis algorithms, OCL expressions are part of the meta-model itself.

ICDMS adds some useful features to standard OCL:

- Applying to any object: is("type"): it returns true if the object type is "type". It is an alias for oclIsTypeOf
- Applying to ICDObject: from(): it returns the set of entities that depend on it, parent(): it returns the parent of it (the object where it is defined in)
- Applying to string: match("expression"): it evaluates matching with regular expressions, toInteger(), toReal(), toLower(), toUpper(),.
- Applying to Real: abs(), exponential(“power”), round(), floor().
- Applying to Integer: abs(), div(), mod()

2.5 Template engines

A template engine is software or a software component that is designed to combine one or more templates with a data model to produce one or more result documents. A template processor is used in Template Oriented Programming. A result document is any kind of formatted output, including documents, web pages, or source code …

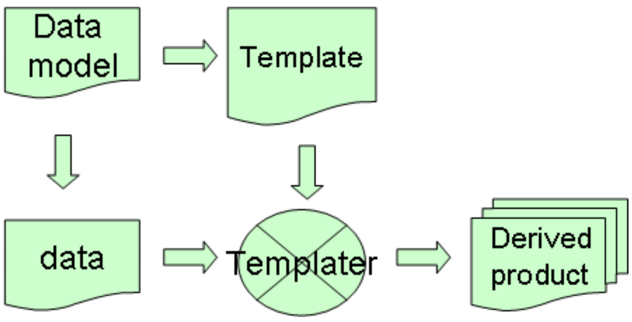

Figure 2: Template engine

The usage of template engines provides some benefits:

- encourages the usage of patterns for design and coding, increases the degree of reuse and reduces the probability of errors introduction
- enhances productivity by reducing unnecessary reproduction of effort (Automatic product generation)
- enhances teamwork by allowing separation of work based on skill-set (e.g. … application domain vs. deployment domain). (Role specialization: Each one do what it better knows)

The usage of templates provides some benefits:

- Facilitate implementation and documentation of patterns
- Set bounds to error introduction

There are some circumstances that make a scenario more suitable to be managed with templates:

- A simple pattern for the solution can be found
- The ratio between the effort to develop and run the template and the effort to develop the product is low or very low
- The problem is typified
- It is sure or foreseeable that the same mistake will appear many times

2.5 XMI/XSLT/XSL-FO

The **XML Metadata Interchange** (**XMI**) is an standard for exchanging metadata information via Extensible Mark-up Language (XML). It can be used for any metadata whose meta-model can be expressed in Meta-Object Facility (MOF).

The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages. In fact, this is the format used by EMF editors to support data persistency for Ecore models.

XSLT (**XSL Transformations**) is a declarative, XML-based language used for the transformation of XML documents into other XML documents.

For parsing XSL, we use MSXML with some jscript extensions to make easier the XSL template design.

XSLT is chosen for XML to XML transformations.

XSL-FO is a facet of XSL, specialized in formatting XML data. It is chosen as a basis for ICD documentation generation.

2.6 Smarty

Smarty is a template engine for PHP. It is chosen for XML to no-XML transformations (source code …).

A special XML parser (written in XSL and PHP) has been developed to provide PHP input data to Smarty from XMI.

2.7 **Altova® StyleVision®**

It is an application for graphically designing and editing proprietary style-sheet (SPS) that can be used for generating XSLT style-sheets based on the SPS design (Both XSLT 1.0 and XSLT 2.0 are supported.). The XSLT style-sheets can be used outside StyleVision to transform XML documents into outputs such as HTML, PDF and RTF.

3. ICDMS Architecture

ICDMS comprises two complementary aspects: ICD data management and ICD data transformations. They are connected through the XMI representation of ICD data.

3.1 ICD data management

One of the key aspects of this - open source based – solution is the integration with DOORS.

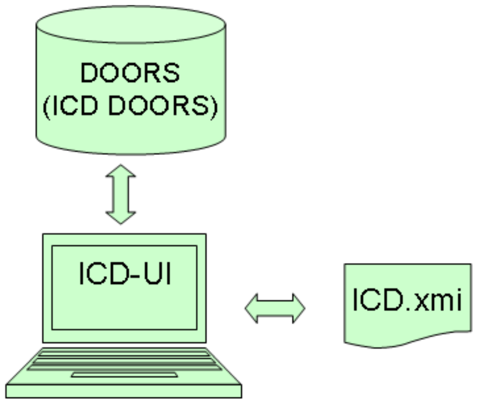

Figure 3: ICDMS Architecture. Data Management

The basic way to interconnect ICD-UI is invoking DOORS-DXL script execution through the available DOORS-COM interface, combined with the usage of simple XML temporary interchange files.

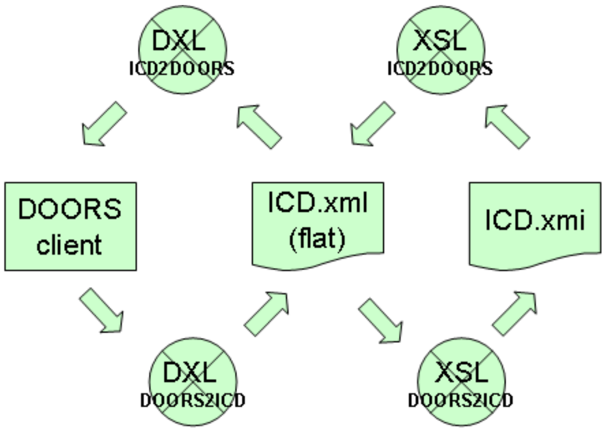

Figure 4: Data transference. ICD-UI <=> DOORS

Moreover, because of some poor performance aspects of DOORS, most of the processing has been allocated to ICD-UI, in order to the keep the operation time response within acceptable margins for user-interface based applications.

Anyway, the mechanisms to connect and to synchronize the front-end with DOORS have been designed to guarantee the robustness of session protocol.

3.1.2 ICD User interface (ICD-UI)

The front-end application (ICD-UI) has been developed with EMF, based on the Ecore meta-model and OCL analysis, following principles MDA.

The manual implementation of ICD-UI features has been undertaken by means of generic programming techniques (EMF reflective API) and user preserving sections in order to be as independent as possible from the details of the meta-model and the consequent EMF automatic code generation.

The programming language is Java.

Available Eclipse plug-ins have been considered, and used when appropriate.

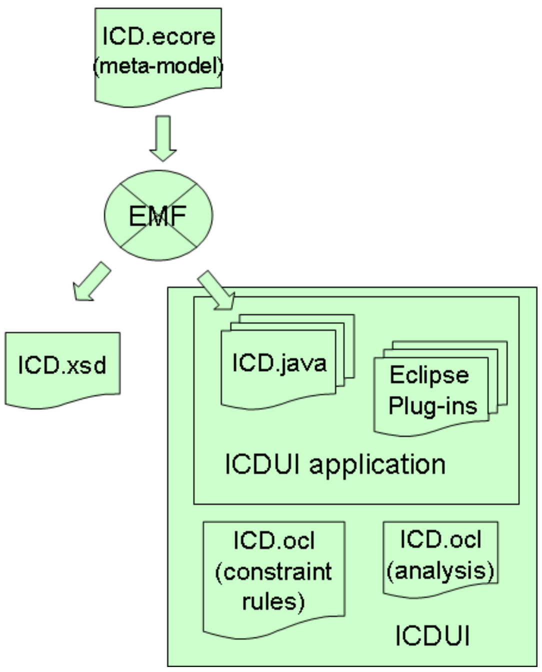

Figure 5: ICD-UI Building

3.2 ICD data transformations

The nature of ICD (massive and strongly typified information) and of its usage (many and heterogeneous users -system/software development, test benches configuration, on-ground/flight tests-; and reduced number of transformation patterns) makes it very suitable to be managed by means of automatic transformations, that allow to put the effort on the design of generic and reusable patterns.

The template engines used so far are MSXSL for XML to XML transformations (XSLT), and Smarty for others.

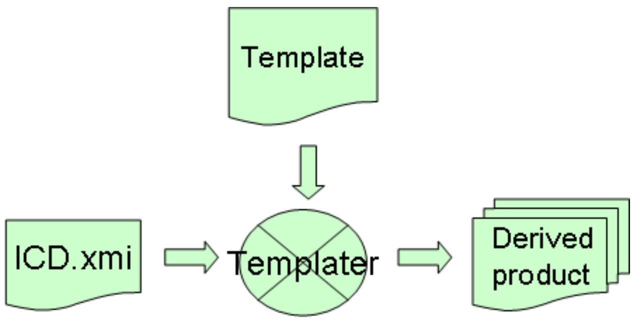

Figure 6: ICDMS Architecture. Transformations

4. ICD Meta-model

For the elaboration of ICD meta-model, previous knowledge of traditional ICD tools (ALBATROS, EPT-ICD, eICD …) have been taken into account.

The structural aspects of the meta-model for ICD (entities, attributes, relationships, cardinalities …) are written in Ecore, and they are mostly based on the entity-relationship paradigm. Additionally, the specializations are described by means of inheritance relationships, following an object-oriented approach.

The top-level container entity is the Module that represents the unit of information under configuration control. A module can depend on another for its definition, but the dependency cannot be circular.

There is a section for generic ICD information and others for the description of specific aspects of different types of communications.

Inside the generic section, there are element intended to describe data layout (message, signals ...), data scaling and communication architecture (systems, subsystems and networks).

The specifics sections refine the definitions made on the generic one, using an extension mechanism based on inheritance relationships.

4.1 Entity-Relationship paradigm

Entity-Relationship model is used to define ICD meta-model in order to enable normalised and flexible ICD descriptions that avoid the redundancy and facilitate the reuse.

4.1.1 Entity

This class is the base for any entity defined in the meta-model. An entity is the representation of a concept, defined by its attributes, which represents the inherent characteristics of the element, those that it has derived from its own nature.

An entity has independent existence and is uniquely identifiable.

4.1.2 Relationship

A relationship is the representation of a unidirectional tie between two entities. The relationship defines attributes that only exist as features of the relationship, so such attributes are not inherent to the related entities.
A relation can be solved by reference or by containment (that is called in-site declaration).
In the first case, the referenced element is defined in some other part of the model. The reference "with" (not containment) of the specialized relationship is used.
In the second case, the referenced element is defined inside the relationship. This is useful, and facilitates the management of the model, when the referenced element is only relevant for the entity that

references him. The reference "owns" (containment) of the specialized relationship is used.

4.2 Data Layout

A ***Structured*** class represents a data layout definition composed by a set of lower level information elements arranged over a bit stream. For each component, the start bit and the cardinality (in the case of a fixed length array of elements) is specified.

The ***Message*** is the highest level of data layout definition.

The ***Signal*** is the lowest level of data layout definition. This element carries the information.

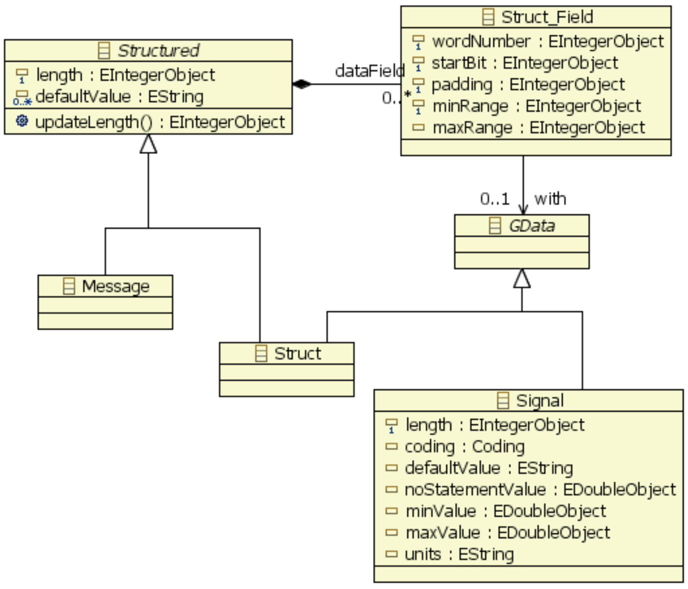

Figure 7: ICD meta-model. Data layout definition

There are some constructs intended to support the description of scenarios for special data layout definition.

Header is a class that allows defining a set of alternative data, with a common part. The concrete data occurrence is indicated by a discriminator field, allocated in the common part.

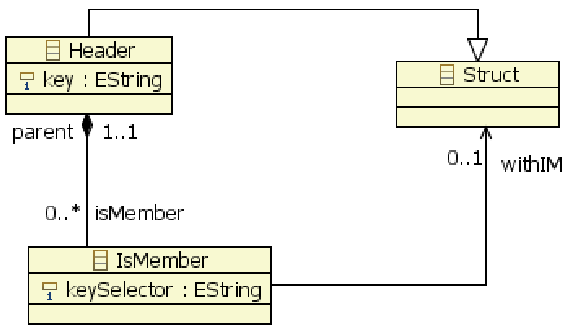

Figure 8: ICD meta-model. Alternative data

Variable-Array is a class that allows defining a list of elements of the same type associated to a counter field. The value of the associated counter, allocated in the same message, defines the length of the concrete list.

4.3 Data Coding

ICD meta-model supports the definition of the rules to code data. In the case of basic coding, it is declared by the value of property *coding* of *Signal* class. This property is a tagged type that includes a number of well defined rule identifiers: *twoComplement*, *TNOctal*, *TNAlphaOctal*, *BCD*, *IEEE754*, *UTF7*, *UTF8*, *UTF16*, *ASCII*, *ISO_8859_1*, *BAM*.

There are other meta-model elements, which can be associated to *Signal* class, intended to define other coding rules:

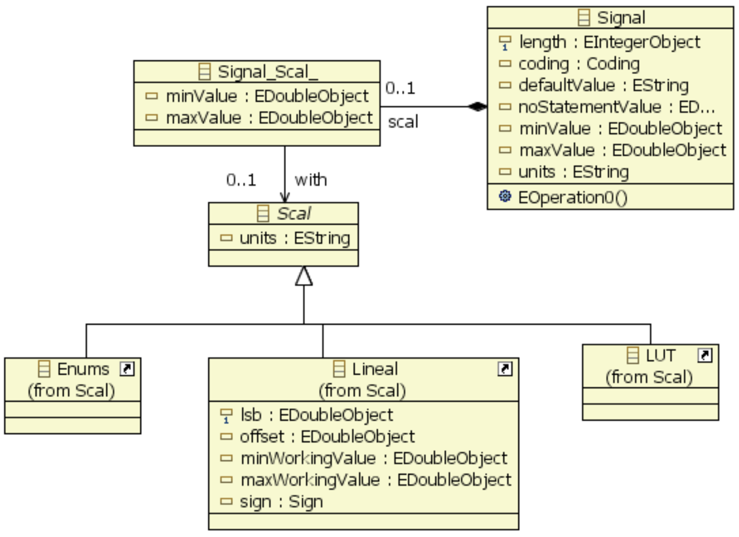

Figure 9: ICD meta-model. Signal scaling

Lineal is a class that allows to associating a lineal transformation to an integer data. The slope of the transformation can be expressed by means of the *lsb* (less significant bit weigh) property value, when definition focuses on data accuracy, or by means of *msb* (most significant bit weigh) property value, when definition focuses on the bottom scale value.

Enum is a class that allows associating a tag set to an integer data.

LUT is a class that defines a data coding by aggregation of a number of non-overlapped value ranges with associated Lineal-like scaling.

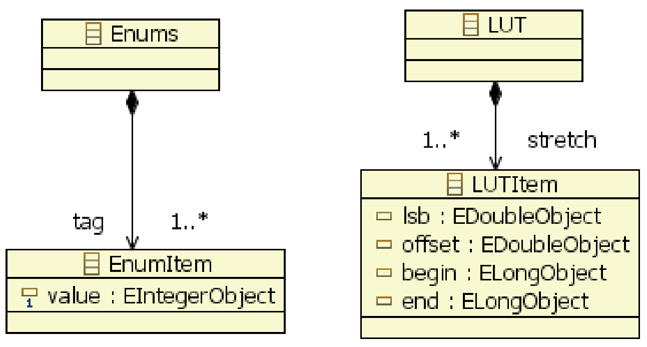

Figure 10: ICD meta-model. Complex scaling

4.4 Communication Architecture

ICD meta-model allows the definition of communication structures with different level of complexity:

Bus is an aggregation of *Message Relationships*. All the *Message Relationships* in the same *Bus* share the same addressing space.

Network allows to link a number of *Equipments* that interchange the messages related from a *Bus*.

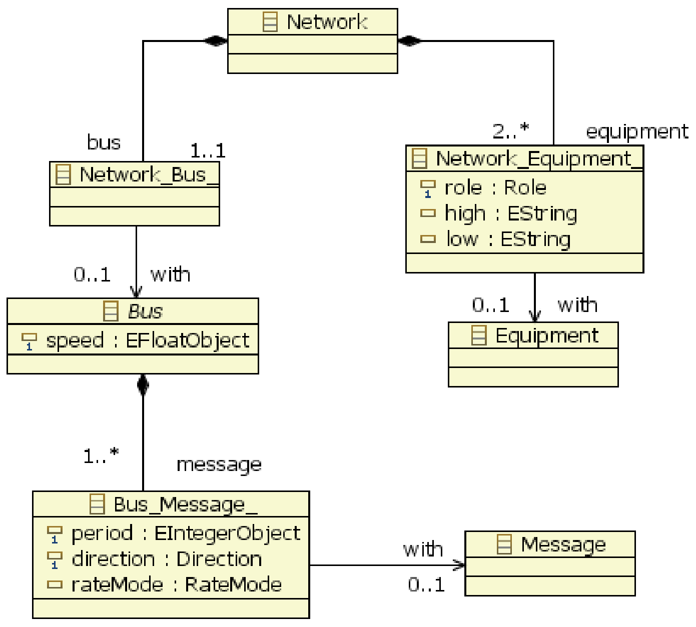

Figure 11: ICD meta-model. Network

System allows to aggregate *Networks* and other *Systems* in order to define elements that are more complex.

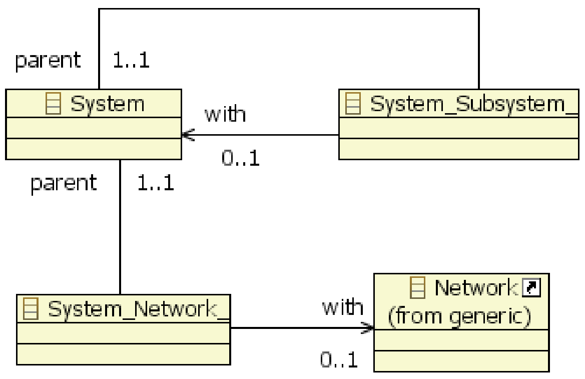

Figure 12: ICD meta-model. System

4.5 Specific packages

There are packages that include the specificities of concrete types of ICD. Following is the list of ICD supported so far:

- ARINC 429,
- MIL-STD 1553,
- AFDX,
- ARINC 653,
- Ethernet
- Analogue and Discrete signals.

5. ICD-UI features

ICD-UI provides user with access to all ICDMS features. It means that user does not need work directly with DOORS user interface.

User friendliness tries to be one of the main strengths of ICDMS. Because of that, we have put extreme care to develop an user graphical interface intuitive, easy to use, and familiar for most of the addressed users, avoiding heavy procedures based on series of forms.

Instead, we have chosen an interface based on trees and tabular windows, that provides mechanisms for fast edition (short-cut, copy-paste, drag-drop) and ICD data overview.

5.1 Browser pane

- This element (the top-right one) contains two tabs: the proper ICD Browser window,
- and a Progress View window.

The first element in the ICD Browser window is the tree of entities inside the selected ICD module. Each tree node can be expanded or collapsed.
The selected module is the selected one in the Views pane (the left one).

Furthermore, all modules on which the selected one depends, either directly or indirectly, are showed by means of additional trees.

The Progress View window shows the execution status of tasks that are running in background.

5.2 Properties pane

This element contains the proper Properties window (on the bottom right), and sometimes, other optional windows (Console).

The Properties pane gives access to the value of the attributes of the selected element for edition.

The "Selected element" is the last one that has been selected either in ICD Browser or in Views pane.

5.3 Views pane

This element contains several tabs. Two of them (General and Message) have a common behaviour on configuration capabilities.

Since both are based on a tabular window, where rows represent a set of elements, and columns represent the element properties, both take advantage of the same filtering and ordering features.

There is a status line, on the top of Views pane, intended to show some information related with the active Module like write permission, synchronization with Database status or Control Configuration data (Baseline identification).

5.3.1 Search view

This is a complex view, composed of two parts:

- a query console and
- a search result window.

The search result window is a tree-table window that shows the findings of the last query. It allows inspecting both the backwards and forward dependencies of found items.

5.3.2 Information view

This element is intended to log the results of checking and analysis of ICD. It is a reactive window that allows to select in the Browser the entity affected by each information entry just clicking on it.

5.3.3 Problem view

This element is intended to show any kind of error reports.

It is opened automatically when a problem occurs while loading a model.

6. ICDMS Features

6.1 Module management

Modularity is a fundamental aspect of ICD description that affects the performance of the deployment (size of DOORS formal modules affects the operation response time) and mainly on the reusability. It has been carefully taken into account in the meta-model design.

A Module is an information assembly that is a configuration control unit (see ICD meta-model description).

Each Module under configuration control ("**controlled**") is stored in a corresponding DOORS Formal Module. Besides, either "controlled" or not, all Module are stored in a XMI file (EMF feature).

It is possible to build a secure XMI file (**controlled files**) for controlled Modules. These files include a checksum that allows detecting undesirable changes in the content. These files are useful to sharing ICD information without the need of using DOORS.

Obviously, controlled files are not suitable for ICD edition.

6.1.1 Module dependencies

ICD Meta-model defines how ICD can be described by means of a set of related modules.
Each Module can depend (always explicitly) on other controlled modules, but circular dependencies are forbidden.

Each referenced module is loaded automatically as a resource (additional tree into the Browser Window).

A frozen version of a module (baseline) cannot depend on a no frozen one.

There cannot be dependencies between elements into Modules that are not directly dependent.

A Module dependency cannot be destroyed while there is some relationship between their elements.

6.1. Load a Module

It is possible to load controlled or uncontrolled Modules (Open / Open checked module).

A controlled Module can be loaded either from Database of from a controlled file.

When loading a Module from Database, It can be selected a Baseline version or the "current" one (the last working version, not frozen).

6.2 ICD edition

6.2.1 Write permission

When working with an uncontrolled Module, the permissions are the same as for the underlying XMI file. The user can manage it with file system. In that case, ICDMS is not responsible about concurrent edition.

When working with a controlled file, it is forbidden to modify it. Anyway, the user could do it changing the XMI file properties, but the file would lose its "unmodified" condition so, it will not be any more a correct controlled file.

When working with a controlled Module from Database, The menu option (lock/unlock) can be used to switch the write permissions. Only one user each time is granted with writing rights.

This mechanism, coordinated with DOORS, prevents form collisions while editing a module.

6.2.2 Create Element

All the elements belong to a containment hierarchy that starts with a Module element. A new element can only be created below an existing one.

The creation of a new element is done by calling the "New Child" or "New Sibling" options of the contextual menu whereas the parent or sibling entity is currently selected.

With respect to the selected element, only the adequate types of element, according to ICD Meta-model, can be created.

It must be taken into account that other constraints not defined in Ecore meta-model, but in OCL constraints definition, can apply, but these are not controlled during the creation phase.

6.2.3 Modify Properties

The attributes of the selected element can be read and modified in the Properties Pane. Some of the constraints for the attribute values are declared in the Ecore Meta-model definition and are managed by the Properties Pane.

The "with" attribute of relationships is quite special. It represents the relation with another existing entity.

When "with" attribute is selected for change, the Property Pane provides a list with all the entities suitable to be selected, extracted from the own Module and from the Modules on which it depends directly. Alternatively, the related entity can be dragged and dropped over the relationship.

6.2.4 Delete Element

When deleting a reference to an entity, the referenced entity is not deleted.
When deleting an entity, the references to the entity are also deleted.

The user is not warned when deleting a referenced entity, nevertheless, that condition is shown by means of a colour code for the nodes in the Browser view tree.

6.2.5 Copy or Clone

Strictly speaking, no copies of element can be done, because the element identifier value must be unique. So copies become on a clone action. A new element with different identifier but with similar characteristics is created.

There are two procedures to clone an element:

- Copy the element and paste it in the desired/allowed location
- Drag and drop (while pressing Control key) to the desired/allowed location

Note that when dragging and dropping an entity to a relationship, the entity is not cloned, but a reference is created inside the target relationship.

6.3 Configuration control

Configuration control is delegated to DOORS. It is possible to request all the actions related with this feature from ICD-UI.

6.3.1 Create Module in Data Base

The uncontrolled Module loaded will be introduced into the Database.

A new DOORS formal module will be created in the specified path, with the same name as the Module and with equivalent contents.

6.3.2 Create Baseline

This action allows creating a DOORS baseline for a module. It is possible to define a suffix for the baseline tag and to introduce a comment.

6.3.3 Create Controlled File

This action allows creating a controlled file from the loaded Module.

6.4 Change tracking

We have decided not to use DOORS-CPS for change management directly, because its working way does not match with the problematic of ICD change. Nevertheless, change tracking is based on module and object history DOORS features, complemented with some extensions to manage change request identifiers.

It is assumed that the management of change control (proposals, discussion, planning ...) is done outside ICDMS.

Once a change has been planned and so, it must be implemented, the affected controlled Module must be loaded and unlocked.

For each identified change, a change session will be opened. The menu option Change Request (init-end) can be used to switch change request session status (opened-closed).

The user is invited to introduce the change request identifier (free format). All changes made in the module from this moment until the session is closed, are stored in the History of DOORS, associated to the change request identifier.

The change session can be aborted at anytime. In that case, all the ongoing modifications will be cancelled.

Each time a change session is closed, the changes are stored in the Database and a comment, associated to the change request session, can be introduced.

6.5 ICD Verification

All the constraints that apply to ICD data are defined formally either in the ICD meta-model, or in the associated constraints (OCL) file.

There are three different verification points, where different checks are performed :

- loading the model,
- running "Validate" command

The first verification of ICD model is done when it is loaded.

A check about correctness of the loaded file (XMI) is performed against the XML schema that has been derived form the ICD meta-model.

If there is any problem, the model is loaded incompletely, and the problems messages are shown in the Problems View.

This kind of problems should not happen frequently because they are due to a bad file construction.

Nevertheless, it can be common while tuning procedures to import external data into ICDMS.

Some ICD model constraints derived from ICD meta-model are not reflected in the associated XML schema so, they cannot be detected when loading the model, and cannot be automatically avoided while editing the model. These validations must be requested explicitly, and the result is shown in the Information View.

6.6 ICD Analysis

It is possible to do some automatic analysis over ICD data. This analysis is intended to provide useful information for the system/ICD designer.

The information provided is of the following nature:

- Transmission load of physical resources
- Process load of equipments/subsystems
- Growth capability (message addressing, payload, processing, transmission, connectivity ...) …

The analysis must be requested explicitly, and the result is shown in the Information View.

6.7 Impact analysis

Additionally to the automatic load analysis, it is possible to undertake the impact analysis with the help of ICDMS.

There are two main ways to do that at different level of detail:

- By inspection of inter-Module dependencies
- By using the capabilities of Search Window

For the finest dependency analysis, first select in the Search Window, the entities under inspection. Then select "reverse" mode to get the set of entities dependent on those.

6.8 Queries

It is possible to make sophisticated queries over ICD model data. The result of such queries is showed in "Search View" window. The syntax to build the queries is OCL. It is recommended to get a good knowledge on it. Moreover, it is encouraged to get a good knowledge about the ICD meta-model to build the queries.

6.9 ICD data transformations

So far, we have exercised transformation for a series of derived products:

- Source code for Input/Output (C++ and Ada)
- Test bench (SEAS) configuration files (XML)
- Source code for Test case stimulus/acquisition implementation (Ada)
- ICD documentation (HTML)
- Flight test instrumentation configuration

Nevertheless, probably the list is not complete and other suitable transformations can be found (e.g. bus analyzer configuration).

6.10 ICD documentation

The documentation of ICD data for publishing is produced as the result of an special ICD data transformation, where the input is the ICD model (XMI), an the template is written in XSL-FO.

Because of the difficulties of writing XSL-FO transformations for complex document formats, we have considered the convenience of using a complementary tool to support that task.

After a light assessment of ALTOVA StyleVision and TOPCASE GenDoc, we have chosen the first one because of his greater maturity.

7. Return of experience

The Pilot cases successfully using this ICDMS are:

- the Aircraft Management Information System of the A330-MRTT for the Ethernet data bus interconnection in the code generation and the test benches configuration, and
- the Boom Control and Computing System and the Multi-Function and Control Display of the ARBS in ARINC-429, CANbus, MIL-STD-1553, Analog and Discrete signals, for the code generation, the test benches configuration, and flight tests.

8. Conclusion

In this paper, we have presented the current state of the ICDMS. It should be stressed that many design decisions have been made based on previous usage of ICD tools in the area of data structure.

The innovating issues are based on incorporating new approaches (MDA) and mature tools like DOORS, Eclipse, EMF, Ecore, XML, XSLT, MOF, OCL, … Those have provided features like: tool construction from a meta model, GUI interface in tabular format including edition, check and search features based on OCL, ICDs handling in the same way as requirements (traceability and impact analysis), reuse of ICDs to deal with the product line topic, data model loosely coupled to the output data transformations, templater based approach for transformations.

These features provide a highly scalable and configurable tool to specific domains even different than the aeronautical.

A future extension of the capabilities of this platform could be the integration with UML design capabilities (already available on Eclipse/TOPCASED) to complement the ICD static description with protocol information, described in terms of state-machine, and using ICD data (messages) as protocol data units.

9. Acknowledgement

Special thanks to the EADS Research & Technology network (GIN4 - RTG14) by its contribution inside the OBSYS demonstrator, the technology room where the ICDMS grew up.

Also mention this work was supported by active participation of the AIRBUS Military organisation by providing a lot of comments to improve the tool. In addition, we wish to acknowledge our colleagues in the EADS' Military Aircraft Systems who encourage us to extend the ICDMS capabilities.

10. Bibliography

[1] Anneke Kleppe, Jos Warmer, Wim Bast: "*MDA Explained: The Model Driven Architecture: Practice and Promise*", Addison Wesley, 2003.

[2] Jos Warmer, Anneke Kleppe: "*The Object Constraint Language. Getting Your Models Ready for MDA*", Addison Wesley, 2003.

[3] Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, Pat McCarthy: "*The Java Developer's Guide to ECLIPSE*", Second Edition, Addison Wesley.

[4] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, Timothy J. Grose: "*Eclipse Modeling Framework*", Addison Wesley, 2003.

[5] John Robert Gardner, Zarella L. Rendon: "*XSLT and XPATH: A Guide to XML Transformations*", Prentice Hall, 2001.

[6] Peter Pin-Shan Chen: *"The Entity-Relationship Model: Toward a Unified View of Data"*, ACM Transactions on Database Systems , 1976

11. Glossary

AFDX	Avionics Full-Duplex Switched Ethernet
ARBS	Advanced Refuelling BOOM System
ARINC	Aeronautical Radio, Incorporated
CAN	Controller–Area Network bus
COM	Component Object Model
DXL	DOORS extension language
EADS	European Aeronautic Defence and Space
EMF	Eclipse Modelling Framework
GUI	Graphical User Interface
HW	Hardware
IBM	International Business Machine
ICD	Interface Control Document/Description
ICDMS	ICD Management System
ICD-UI	ICD User interface
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
LUT	Look Up Table
MDA	Model-Driven Architecture
MOF	Meta-Object Facility
MRTT	Multi Role Tanker Transport
MSXML	Microsoft XML Core Services
OCL	Object Constraint Language
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
SLOC	Source Lines Of Code
SW	Software
SEAS	Stimulation, Acquisition and Simulation System
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSL-FO	eXtensible Stylesheet Language Formatting Objects
XSLT	Extensible Stylesheet Language Transformations