Click here to view PowerPoint presentation; Press Esc to exit

How to Make an Effective Information Exchange Data Model

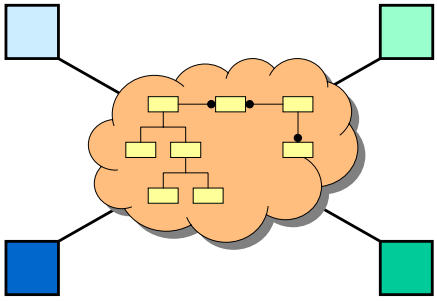
or

The Good and Bad Aspects of the NATO JC3IEDM

Eddie Lasschuyt, MSc
Marcel van Hekken, MSc
Willem Treurniet, MSc
Marc Visser, MSc
TNO Physics and Electronics Laboratory
P.O. Box 96864
2509 JG The Hague
THE NETHERLANDS
T +31-703740208, F +31-703740652

Lasschuyt@fel.tno.nl

2 September 2004[*]



Abstract

Coalition-wide interoperability implies, among other things, that a large amount of C4I systems are able to seamlessly exchange information. A bare necessity for this is to have common exchange languages that unambiguously define the information which is shared among the coalition parties. Information Exchange Data Models (IEDMs) are used for that. But developing an IEDM is a difficult process. The resulting model is often very big and complex, which makes it hard to comprehend and implement. This paper discusses a set of guidelines which help to make better IEDMs. The Joint C3 IEDM from MIP and NDAG is used as a case, being the most mature model of its kind in the NATO C3 environment.

[*] This paper is presented at the NATO RTO IST-042 Symposium on "Coalition C4ISR Architectures and Information Exchange Capabilities", in The Hague, The Netherlands, 27-28 September 2004. RTO is the NATO Research and Technology Organisation; IST is the Information Systems Technology panel of RTO.

Paper presented at the RTO IST Symposium on "Coalition C4ISR Architectures and Information Exchange Capabilities", held in The Hague, The Netherlands, 27-28 September 2004, and published in RTO-MP-IST-042.

Contents

1. Introduction

1.1. Context

1.1.1. Interoperability in general

With respect to information management, large organisations suffer similar problems nowadays. Many heterogeneous information systems support the business processes. Their diversity is among other things manifested by differences in:

- functionality, related to the purpose of systems and to their user groups;
- the kind and format of the information being managed by the systems;
- platforms, including hardware, operating systems and (COTS) applications.

The 'natural history' of information management within organisations has lead to a situation where most of their systems have become 'stove pipes', working independently from each other. There is, however, an increasing need for global interoperability among these information systems, both organisation-internally and between different organisations. Connecting systems at information level is generally advantageous on two aspects:

- effectiveness, for example in terms of increased situational awareness (more accurate and actual knowledge about what's happening at any level of an organisation) or derivation of new knowledge by combining information;
- efficiency, for example by sharing information among systems (and users) which is already present in some system.

The more collaboration there is between departments and with external organisations, the more need there will be to make information systems interoperable.

1.1.2. Interoperability in NATO

In NATO[1] the situation described above occurs on a very large scale. The "Consultation, Command and Control" (C3) process involves many dissimilar parties, differentiating in several ways:

- organisations from different (NATO and non-NATO) *nations*;
- different *kinds* of organisations, such as NATO bodies, military units, NGOs, commercial organisations (such as weather stations and logistic support) and the media;
- military units and headquarters from different *Services*, at different *levels* of command.

All these organisations bring their own information systems, resulting in a varied system landscape with NATO systems, national systems, non-governmental systems, etc. Furthermore, the C3 process encompasses many functional areas, each supported by a variety of C3 Communication and Information (C4I) systems. All these systems have a very dissimilar character in that they have different functionality, handle different kinds of information with different formats and are made by different manufactures.

The existence and diversity of so many systems in NATO C3 context is a fact of life which must be dealt with. Nonetheless, NATO's policy is to achieve interoperability among all major systems. This is driven by the combined/joint and flexible nature of today's operations, that require extensive co-operation between the many participating (military and non-military) organisations. Related to this, *network-centric* concepts also imply broad interaction between systems.

So, some sort of *overall* interoperability capacity is needed in NATO, enabling each system to potentially exchange the most essential information it has in common with any other system. This is a real challenge.

[1] Although NATO is used as illustrative context for this paper, most statements and examples are equally valid in a (wider) multinational/coalition context of military C3.

1.1.3. IEDMs

Enterprise-wide 'system interoperability' implies [1] the automated exchange and interpretation of structured digital information among many heterogeneous systems. An essential prerequisite for this is the *standardisation* of shared information. Early efforts in NATO context resulted in standards such as ADat-P3, which defines formatted messages for C2-related information, but nowadays information standards are mostly specified by means of *data models*. In case a data model is primarily meant for interoperability purposes, it is called an *"Information Exchange Data Model" (IEDM)*.

1.2. Problem description

IEDMs tend to become very complex. This causes that these models:

- are very hard to comprehend by non-modellers;
- result in difficult implementations of database and applications;
- bring about complicated data sets which are difficult to manage and process;
- are only being used partly in practice.

Some other common problems in IEDMs are:

- the use of unpractical modelling constructs;
- the weak relation with unstructured information;
- the difficulty to maintain and extend the model;
- the difficulty to compare (and translate) with other models.

These points embody the major reasons why IEDMs can be hard to use in reality.

1.3. Objective and scope

This paper contains guidelines for making better IEDMs. It describes our opinion on how a well-designed exchange language should look like and what it should include. Applying the guidelines could lead to more effective models, that better fit their purpose. Also, it may make the design process more efficient.

The "Joint Consultation, Command and Control Information Exchange Data Model" (JC3IEDM) is used as a 'case' in this paper. It is the most significant and mature IEDM (being developed) in NATO at the moment. Its long existence makes it an excellent example to learn from, both in positive and negative sense. But in principle the guidelines are applicable to any IEDM.

The paper does *not* contain a complete overview on modelling of information exchange data models. Instead, it is a collection of experiences and best practices in that area, gained from many years of data model developments in several projects, among which ATCCIS/MIP. The paper is intended to be used as a basis for a NATO-wide discussion about developing high-quality IEDMs. It is a sequel to an earlier paper about "Information Interoperability Domains" [1], which describes an approach to achieve enterprise-wide interoperability within large organisations (such as NATO) by managing information standardisation in a centralised and structural manner.

The paper is specifically meant for people involved in data modelling and interoperability. The readers are also supposed to possess some basic knowledge about NATO C3 in general and ATCCIS/MIP in particular. Understanding of the JC3IEDM (i.e. C2IEDM) is convenient, but not obligatory.

1.4. Overview

Chapter 2 gives a short introduction to the JC3IEDM. Chapters 3, 4 and 5 contain the IEDM guidelines, split up in three categories: general, on structural aspects and for multiple models. Chapter 6 summarises the conclusions.

2. The Joint C3 IEDM

2.1. Introduction

The most important IEDM being developed within NATO at the moment is the "Joint Consultation, Command and Control Information Exchange Data Model" (JC3IEDM). It is developed and maintained by the Multilateral Interoperability Programme (MIP) and the NATO Data Administration Group (NDAG). The aim of MIP — and inherently of the JC3IEDM — is "to achieve international interoperability of Command & Control (C2) information systems at all levels from corps to battalion (or lowest appropriate level) in order to support multinational (including NATO), combined and joint operations and the advancement of digitisation in the international arena" [2]. The NDAG is (among other things) responsible for 'data administration' within NATO, i.e. policy and control with respect to the definition of structured information across NATO [6].

2.2. History and scope

ATCCIS, which stands for "Army Tactical Command and Control Information System" was an ad-hoc working group with the objective to develop an international interoperability standard for Land C2 systems. For that purpose, ATCCIS created the so-called "Generic Hub" data model in the early nineties. This model defined the general Land C2 concepts and served as a core (hub) for unifying specific functional area extensions (such as Fire Support, Engineering and Intelligence), to be added in the future. Only a few of these submodels were actually produced. The Generic Hub evolved to the "Land C2 Information Exchange Data Model" (LC2IEDM). In 2002 ATCCIS ended, but its products (and experts) were taken over by MIP. The model was renamed to "C2IEDM" and further enhanced. This includes the addition of a few Air and Maritime concepts for Land-based operation that require joint information exchange. Both ATCCIS and MIP were/are no official NATO programmes, but 'NATO-endorsed'.

Another line of development was the NATO Corporate Data Model (NCDM), under responsibility of the NDAG. This was envisioned to become a suite of closely related data models supporting NATO data administration, including long-term data management and identifying core data structures within the alliance [6]. The NCDM would be a kind of dictionary, containing the semantic definitions of structured NATO C2 information. It was *not* intended to support information exchange. The NCDM was to consist of a single Reference Model and various View Models. The Reference Model would be the core, containing the most common, enterprise-wide concepts. A View Model would contain a more detailed data definition, specific for some user community. The View Models re-use the common concepts from the Reference Model and are in that manner related to each other. Only one View Model has been submitted and incorporated in the NCDM: the MIP C2IEDM. Some other 'candidate' View Models exist as well, for example on Airspace Control, Maritime Mine Warfare and NBC.

In the beginning of 2004, the MIP C2IEDM (Edition 6.1) has been merged with the NCDM Reference Model (Version 4.0.0) and is now called the JC3IEDM [2]. The JC3IEDM is under development. Its (future) scope will be 'upgraded' from Land-based C2 to *Joint C3*. This will include:

- joint (and combined) information, to support activities in which elements of more than one Service (or nation) participate;
- consultation information, to support the exchange of views and the conduct of deliberations amongst the highest authorities of the Alliance and member nations;
- Service-specific C2 information, at all levels of command.

Notice from the third bullet that this 'joint' model also encompasses information exchange purely within Armies, Navies or Airforces (combined). The JC3IEDM is intended to cover all elemental information in NATO C3 perspective. However, it must be said that although this scope is derived from [2], it is not literally stated in there. The exact eventual scope seems to be not completely clear. It may in the end depend on which Information Exchange Requirements (IERs) are actually offered to MIP/NDAG by other NATO bodies and programmes.

2.3. Current status

Currently, MIP and NDAG work towards this 'all-enclosing' IEDM. New and outstanding IERs are being brought in, either via MIP (Land) or via NDAG (Joint, Air, Sea). Version 1.0 of the JC3IEDM is planned to be available (for testing) in June 2006.

The latest version of the JC3IEDM, available outside MIP, is equal to C2IEDM Edition 6.1 (20 November 2003). The actual analysis described in this paper is therefore done on the C2IEDM. (In fact, it does not really matter which version is used, because it is purely meant as an example). The complete (logical) data model schema can be found at the end of this paper, as a separate sheet. It is also available on the MIP website (http://www.mip-site.org/). The model is drawn according to the modelling/schema technique "IDEF1X" and is made with the ERwin tool.

2.4. Achievements

In general, the JC3IEDM is the best IEDM in NATO. It is the most significant and mature one available. One of the good things about the JC3IEDM is that it has succeeded in standardising a lot of C2-related information in a structural manner. It comprises a very accurate and extensive description of that data.

Also, the whole principle behind ATCCIS/MIP, using a common IEDM to achieve interoperability, is absolutely good. This concept clearly has many advantages over alternative interoperability solutions, such as dedicated interfaces or formatted messages.

The data model has been tested and demonstrated on multiple occasions, which brought a lot of lessons learned and also has increased its value and fame.

2.5. Weaknesses

The JC3IEDM in its current form also has some serious shortcomings. It is very large and complex. This makes it hard to comprehend, implement and maintain. In addition, it has taken almost 15 years to come this far, which is mainly due to a slow decision process. This, in turn, is caused by several factors, among which the large number of nations involved, the models complexity and national issues (legacy systems, commercial interests, politics, etc.). Chapter 3 discusses this, and other general concerns, in more detail.

Furthermore, there are several structural aspects in JC3IEDM which cause certain problems. Among other things, the model is partly too generic or too explicit, is not denormalised at the physical level, contains metadata and static data and is little integrated with unstructured information. For further explanation on these issues, see chapter 4.

Finally, the scope of JC3IEDM, moving towards a single IEDM for the whole NATO C3 area, has become too big. In our view, MIP/NDAG will never be able to manage all activities and parties in such a way that it leads to an IEDM that is practically useable in its full extent. Despite of this, we still think MIP made a good decision in going joint. It has momentum (not much, but still) and it should continue. Bringing all Services together is still a major step in interoperability context. Better do it like this, than not at all. How this whole process could be optimised is written in chapter 5. Notice also that the current scope has only partly been set on purpose, due to historical events (e.g. the recent merger).

2.6. Conclusion

More specific (positive and negative) issues of the JC3IEDM will be mentioned in the next chapters, *primarily to illustrate the guidelines*. Of course, these deliberations could also be used as input for the ongoing development of JC3IEDM.

3. General Guidelines for IEDMs

3.1. Introduction

This chapter describes the more general guidelines that apply to an IEDM, for instance about its scope, without diving into specific data model structures (which is done in chapter 4). First, some introductory remarks are made to set the context more precisely.

3.1.1. Information interoperability and standardisation

Heterogeneous systems use different (internal) data structures. Interoperability between diverse systems requires (among other things) the definition of a mutual information standard [1]. This enables systems to equally interpret (digital, structured) information, which has been exchanged between them. Without a common understanding of the information, systems will never be able to interpret it in the same way. When a British C4I system reports a hostile unit to a German C4I system, both should have equal comprehension on the unit's size, location, status, etc. They must, so to say, 'speak the same language'. A *common exchange language*, or 'Esperanto', is needed. It has to be standardised among all players (systems) that want to interact (see figure 1). This information standard defines the *semantics* (what it means), the *syntax* (or grammar, how it is structured) and the *lexicography* (how it is represented) of the information to be exchanged. Since information standards are commonly defined with data models, exchange languages are also called *Information Exchange Data Models (IEDMs)*.

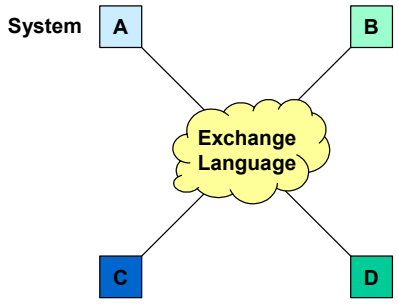


Figure 1 — The common exchange language

The contents of an exchange standard is based upon the *Information Exchange Requirements (IERs)* that exist between the organisations behind the involved systems. IERs basically describe 'who needs what from who' and are the result of an information analysis of some field of interest.

3.1.2. Information exchange

This paper focuses on the information standard and discusses several aspects of it. Other issues with respect to interoperability will *not* be discussed here. This applies especially to the *method* by which information is being distributed, including the exchange mechanism that transfers information between systems (e.g., database replication, web services) and data management rules that ensure the consistency of shared information. In our context these features are irrelevant, because no matter what exchange method is used, the meaning and structure of the information to be shared still needs to be defined.

3.1.3. IEDM optimisation goals

Modelling guidelines are needed to solve the problems stated in paragraph 1.2. This must lead to optimal IEDMs, that are:

- relatively easy to understand and explain to non-modellers (e.g. policy makers, project managers, future users and application developers);

- relatively easy to use in an implementation (databases, translators, applications);
- structured not very complicated, making creation and processing of information easier and faster;
- able to integrate *un*structured information with structured information;
- capable of being extended with new types of information, by relating them to existing types and without changing these (backwards compatibility);
- rather similar to each other so that different data models can be related easier and translations between their data formats becomes feasible.

The guidelines described in this paper will highly improve data models with respect to these aspects.

3.2. Relational data models

A *data model* is an unambiguous and structured description of the information relevant for a certain business area. It is an *abstraction* of a scoped part of the real world in that it only contains the most essential part of the information being utilised in reality and that it records this information in a simplified and formalised manner. The model specifies the syntax, semantics and lexicography of the information in both a schematic and descriptive manner. Data modelling has become a proven methodology of recording and structuring the results of an information analysis (IERs in our context).

Current data models are mostly of *relational* nature, also called Entity-Relationship Diagrams (ERDs). They represent objects, their properties and associations between objects by means of *entities*, *attributes* and *relations* respectively. Additional restrictive rules about objects, such as allowed property values and possible combinations of property values (business rules) are captured in *domains* and *constraints* respectively. Besides a schematic representation, a data model also encloses textual specifications, among which entity and attribute definitions, domain tables, constraint descriptions and example instances. More explanation about relational data models in general can be found in literature. There are many techniques and tools for drawing data model schemas and documenting data definitions. The modelling and schema technique used in this paper is equal to that of JC3IEDM, namely “IDEF1X” [7,8].

Although a data model is a widely used instrument to structure and unambiguously define information, other methods are available as well. An information standard for *exchange* in particular could also be specified through a glossary, formatted messages, an XML schema, an object model or an ontology. However, a relational data model is considered being the preferred way to specify an information exchange standard. A brief justification:

- Glossaries are collections of terms with definitions. They are poorly structured and insufficient for automated information exchange, because of missing syntax and semantics.
- Predefined sets of formatted messages (such as ADat-P3) are an inefficient way to standardise information (because much data is repeated in multiple messages) and offer little flexibility in selecting subsets of information to be exchanged. Nonetheless, formatted messages are still a good method to exchange information, but in that case an ‘overarching’ data model makes it possible to unambiguously structure the whole set of information inside all the messages.
- The eXtensible Mark-up Language (XML) is in particular useful to define a standardised syntax for documents and messages. It ‘labels’ information, to indicate (for example) that a particular data value “Bordeaux” represents a city (and not a whine). However, some kind of data model is still required to describe the *meaning* and *context* of the information (in this example the attributes of a ‘city’). XML offers schema techniques for that, although these are less mature (regarding method, tools, etc.) than relational data models. At least XML is a valuable instrument to pack information (in a message) when it is exchanged between systems. It could be used in combination with relational data models: XML defines the *message syntax*, the IEDM defines (semantics, syntax and lexicography of) the total set of available information.
- Object models are focussed on objects and their behaviour, not on the information itself. They enable one to specify the allowed mutations (“methods”) on objects. This implies a lot of extra specification

work and is not really essential as part of an information exchange standard, which is primarily aimed at achieving common understanding of shared information. (The behavioural aspect of an object model could be useful as a way to define constraints, though.)

- Ontologies are data models with a more formal and methodological way to express complex constraints. So, ontologies offer more semantic expression than relational data models. The 'semantic richness' enables a more powerful range of reasoning to be conducted on the information. Ontologies are particularly being used for Artificial Intelligence-like applications for functionality such as natural language interpretation, data fusion and automated inferencing. A very strict representation of knowledge is needed for these applications. The drawback of ontologies is that they are (much) more complex than relational data models and require more effort to develop (while defining just the data is already hard enough). And, most importantly, system interoperability does not gain very much by the definition of very complex business rules. It is true one can increase the expressiveness of an exchange language by using an ontology, but without much purpose and at a high price. Data models (including simple constraints) are able to define sufficient semantics of the information being used by and exchanged between the majority of information systems.

In conclusion, we consider relational data models to be a well-suited technique to specify an exchange language. Whenever this paper talks about a data model, we assume it is a relational data model.

3.3. Data model categories

3.3.1. Data model roles

Data models can be used for three different purposes (see figure 2). Depending on its function, a model can play one or more roles. The traditional one is an *application data model*. It is meant to define and structure the data which has to be presented, mutated and stored in a certain information system. The model is used as a basis for the development of applications and databases within that system. Early data models were solely intended for this purpose.

The second role is the *(information) exchange data model* (IEDM), the subject of this paper. It standardises the data which is shared and (potentially) exchanged between heterogeneous systems (i.e. with different application models). Such a common exchange language is the obvious way to achieve interoperability at information level. The applicability of the language is usually limited to a certain information area, such as Logistics, and (implicitly) to a certain group of information systems. The IEDM is used in several ways:

- to support the information analysis for IERs and the standardisation process;
- to describe the meaning of the data so it will be interpreted equally by all players;
- to describe the format and structure of the data actually being exchanged;
- to form a basis for developing data exchange mechanisms (e.g. as replication database).

The third role is the *standardisation data model*. This model acts as a 'structured glossary' that standardises meaning, structure and format of information types which are very common within a certain wide context (enterprise, branch, etc.). A standardisation data model is not meant to be implemented. Instead, it is used as a basis for further modelling. Whenever a large organisation has to deal with several modelling efforts, both for building specific systems (applications, databases) and connecting different systems (interoperability), it is wise to make use of such an 'umbrella' model (see also par. 5.4). The most fundamental concepts of the organisation are formalised and standardised in here. If different application data models and exchange data models are based upon the same standardisation data model, they will incorporate similar concepts in exactly the same way. This reduces overlap in modelling work and increases compatibility between these models, making data easier to be transformed between different formats.

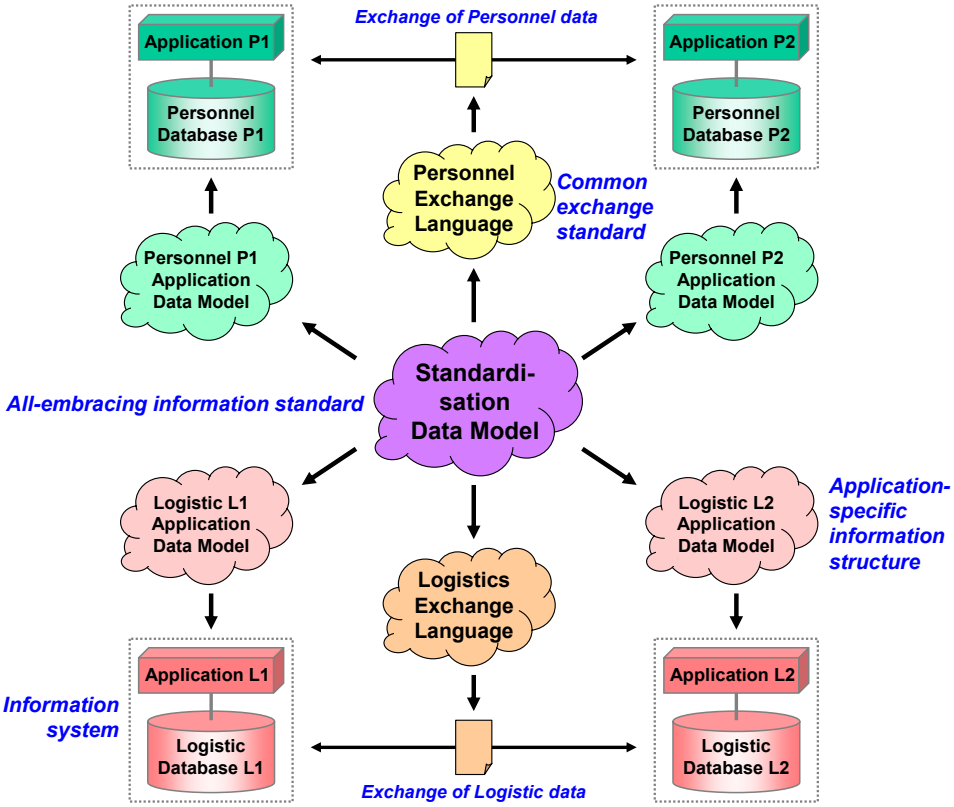


Figure 2 — Different roles for data models

The main guideline here is to be clear about the role of a data model. An IEDM is sometimes confused with an application data model. Some nations, for instance, have implemented their C4I systems directly on the LC2IEDM. This is an exchange model, not an application model. It could be used in both roles, but one should be very cautious here, because the national requirements may differ from the international ones and in this way a nation may have little control over the structure of its own national C4I database.

Also, confusion between the purposes ‘exchange’ and ‘standardisation’ is possible. The recent merger of the LC2IEDM and the NCDM is an example. The first is a pure exchange model, the second was meant to be a standardisation model. It is not quite clear whether the resulting JC3IEDM will only serve data exchange or will be used for wider standardisation purposes as well. In our view, JC3IEDM should support information exchange only. Besides that, an *additional* standardisation model is needed to assure compatibility with other IEDMs (see par. 5.4).

3.3.2. Data model levels

Data models can be specified at different levels, representing distinct construction phases with distinct purposes and consumers. Mostly the levels ‘conceptual’, ‘logical’ and ‘physical’ are used. There are dissimilar interpretations for what these levels actually encompass. In our context of IEDMs we use the following.

First of all, there are no conceptual, logical or physical *data models*. Instead, a data model can be defined and visualised at three levels. We say it contains three (interrelated) *schemas*.

A conceptual schema is a global reflection of a certain information (interoperability) area, meant to clearly state its *scope* and define its *main* information objects (called ‘business objects’). It is a simplified form of the other two schemas, describing the data model as concrete and understandable as possible. In JC3IEDM context it will only show the major entities, attributes and relations (PERSON, ORGANISATION,

MATERIEL, ACTION, etc.). The conceptual schema is mainly used for communication purposes, i.e. to explain function and scope of the model to outsiders.

A logical schema is a complete and normalised (see par. 4.3) variant of the exchange data model. It contains all entities, attributes and relations, including definitions, constraints, explanation and examples. Also logical domain values (e.g. unit status "Operational") are listed. A data model is usually represented and viewed by the logical schema. *If not explicitly stated otherwise, we always refer to the logical schema when talking about a data model.*

A physical schema is the concrete description of the information standard, i.e. what is actually (physically) being exchanged. Among other things it specifies the physical names of entities and attributes ('tables' and 'column' in database terms), the datatypes of attributes and the physical domain values (e.g. unit status "OPER"). The structure of the physical schema may differ from the logical one. For instance, optimisations such as denormalised or redundant entities can be made to increase the performance of implementations of exchange mechanisms (e.g. to enhance retrieval/update speed or to decrease storage space). Unlike the more common understanding of a physical model, i.e. denoting how a logical model is implemented in a certain software solution, our interpretation is still independent of the implementation (database-independent).[2]

All three data model levels are required for a complete IEDM. So, one should standardise on all IEDM aspects described above. Distinction between the levels is very useful. It makes the data model design process easier and more transparent.

To a certain extent, the JC3IEDM has been split up in this manner, which has proven to be very useful. The overview document of JC3IEDM [3] contains a conceptual schema. The JC3IEDM itself and the extensive documentation [4] contain both the logical and physical schemas. MIP does not make a real distinction between the two. Structurally they only differ in the addition of a few meta-attributes (such as owner-id and update-seqnr). It would have been better to optimise the physical schema for exchange.

3.4. Scoping of IEDMs

What should and should not be included in an information exchange standard? Below, several aspects of a proper scope are discussed.

3.4.1. Clear scope and purpose

An IEDM must have a *well-defined* scope. It should be explicit and clear what kind of information is included in (or excluded from) the model. Also the purpose of the model (and implicitly the target user group) should be indicated. This ensures the IEDM will be applied correctly, for what it is meant for. In addition, it will avoid confusion between and/or overlap with other IEDMs.

The scope of the predecessor of the JC3IEDM, the C2IEDM (see par 2.2), is quite apparent [3]. It is basically a corporate view of the data that reflects the multinational IERs for multiple echelons in *land-based* wartime operations and crisis response operations, including *joint interfaces* that support land operations. This is also called a Land "Common Operational Picture" (COP). The data model is focused primarily on the information needs that support the operations planning and execution activities of a military headquarters or command post.

However, the intended scope for the future JC3IEDM is not (yet) completely clear [2], as already said before. It is, for instance, not absolutely sure whether the model will eventually cover the IERs for NATO C3 in total (including air and maritime operations) or be limited to land and joint IERs.

[2] Such an implementation-dependent schema may be useful as well, but is a national/local matter and falls outside the scope of the standardised IEDM.

3.4.2. Limited scope

The scope of an IEDM must be *manageable*. A standard that covers too much different functionality and user groups cannot be developed and maintained efficiently. On the other hand, the IEDMs should not get too small, because that will result in a large number of information standards which undermines the main idea behind standardisation (making as many users as possible to share the same product).

What is the maximum size for an information exchange standard? A general condition is that it remains manageable with respect to overall approval and development. Strict rules for finding the maximum scope are hard to give. *The basic directive is to have a group of systems that exchange much similar information.* In other words, systems that have *high-intensity* and *coherent* information flows in-between, are candidates for using a common exchange language. Normally these systems have comparable functionality. So, an IEDM should contain *highly correlated* information types, often all related to a single subject or functional area (e.g. Logistics, Intelligence, Personnel). An IEDM should not cover a variety of hardly related information types.

Besides this, the organisations that *own* or *use* the information inside the systems should also be correlated. The total set of information enclosed by an IEDM must be owned and exploited by a *limited* number of *related* organisations. Such a 'homogeneous user group' keeps the standard manageable. This would not be the case when there were too many potential owners for a certain type of information and these owners were not organised such that only a few representatives were needed to be involved in the standardisation process.

The JC3IEDM is a good example of an IEDM that has a scope which has become far too wide. Defining a coalition Army C2 standard (the predecessor LC2IEDM) with over 20 nations is already a big challenge, but the current scope (joint/combined) seems hardly doable. It appears one tries to cover too much IERs with this model. This opinion about the over-sized scope is supported by a number of observations:

- The data model is very complex (see par. 3.5) and is specified and explained by means of some extremely thick documents [4].
- Only a few people in the world understand the entire model. This weakens the whole purpose behind the model, unambiguously defining the information which has to be shared among many parties.
- Progress is very slow. No fundamental changes have been applied to the model over the last ten years. This could be interpreted positively, in the sense that the model is apparently very satisfactory and stable. Also, its flexibility may have caused little modifications (see par. 3.6). However, there are enough problems and issues that could be improved, as can be read in this paper. The main reason behind its stability appears to be the fact that changes are difficult to agree on and incorporate, due to the large number of parties involved and all kinds of non-modelling (e.g., political, financial, commercial) interests at the background. This makes evolving the model a time- and resource-consuming business.
- Only a small part of the data model is actually being used during tests and exercises. Although one has tested the ATCCIS/MIP concepts (including the data model) already many times, for large portions of the model still no applications or exchange contracts exist.

Remark: The reasoning above leads to the notion that information exchange within a large organisation could require the use of *multiple* IEDMs. Chapter 5 provides more explanation on this.

3.4.3. Balanced abstraction level

A data model should represent a piece of reality at a certain level of abstraction and detail. If not, the model will become 'unbalanced', making it harder to grasp and maintain. Compared to broad concepts, details require more expertise to fill in and tend to be more unstable. Therefore one should avoid too much variation in level of abstraction and detail.

The JC3IEDM is very generic, but also contains quite some details. There are seven entities about medical facility status and also seven concerning harbours. Such detailed information types do not 'fit' inside this model, given its (wide) scope. This problem is solved by taking two measures:

- some of the details are modelled more generically — medical facility status in FACILITY-STATUS;
- other (more specialised) details are put in another IEDM — medical facility status in a Logistics IEDM and harbour in a Maritime C2 IEDM, see chapter 5.

The last remark does not mean that HARBOUR should be eliminated completely from the model. This entity may be appropriate, just like for instance AIRFIELD, but it has too much specific attributes and related entities that contain most of the details and should thus be removed. This is a typical example of information that is too detailed and makes the model unbalanced.

3.4.4. No static information

An IEDM describes the semantics of the information that can be exchanged between heterogeneous information systems. Information about objects that are more or less static (i.e. may only change over a long period of time) should in principle not be included in an IEDM, because it will only enlarge the model without having much purpose. For instance, encyclopaedic or 'type' information (see also par. 4.7) about an F-16 fighter, like its speed, strike capabilities and weight, is not likely to change very often. Also, new fighter types will not occur every week or so. Another example is a list of nations, which also tends to be reasonably stable.

It is therefore better to describe such information (if at all necessary) in a *separate* data model. It serves as background information, which could be distributed in a different way than the dynamic information of the IEDM, for instance via a yearly update on CD. This is much cheaper than via some (near-)real-time exchange mechanism! In case only references to certain static data items are required (and not the properties of that data), attribute *domains* should be used. (See par. 4.7 and 4.8 for further explanation on this subject.)

The OBJECT-TYPE tree in the JC3IEDM appears to be rather static. It is also very extensive (no less than 34 entities, not including several associative entities). Being able to represent a type (as opposite as to an item) may of course be necessary, but such a bulk of static data seems not really useful to be exchanged along with the other data. It is doubtful whether type information is part of the IERs in the first place. This hesitation is also fed by the fact that JC3IEDM does not include an ACTION-TYPE entity. More general, the model does not contain any doctrinal aspects, although this may be quite relevant information for the C2 decision cycle. One of the reasons it has been left outside, is probably its static nature.

Besides OBJECT-TYPE, a similar reasoning applies to the constructions of ESTABLISHMENT and CAPABILITY in JC3IEDM.

3.5. Simplicity of IEDMs

Enormous 'spaghetti-like' models make nice wall paper in an office, but are hardly useable in a serious way. They tend to be very hard to comprehend and handle. Keeping things as simple as possible is essential for making IEDMs really work in a large environment. An IEDM should be fairly easy to explain and understand (to/by non-modellers), develop and maintain (by a large group of people) and implement (by programmers in databases, translators, etc.). If not, their usage will be less widespread, diminishing the successfulness of the exchange standard.

The complexity of a data model largely depends on the number of entities. The other complicating facets (number of attributes, relations and additional constraints) are usually correlated to that, on a scale which depends on the way of modelling[3]. A large number of entities is mainly caused by:

[3] For example, generic modelling (see par. 4.2) results in less entities, but more constraints.

- a large scope (see par. 3.4.2) and/or
- the urge for a completely normalised data model (see par. 4.3).

To limit the amount of entities in an IEDM one should pay attention to these two issues.

When does an IEDM become too large? An exact number of entities can of course not be given, but in general one should be able to overview the model quite easily. A guideline could be to strive for a logical schema that fits (in moderately readable format) on an A3-sized piece of paper. That would approximately include between 20 and 80 entities. Of course, this highly depend on several circumstances.

The JC3IEDM clearly exceeds these figures. It contains 195 entities, 785 attributes and 277 relations. The accompanying documentation with definitions, domain specifications, etc. takes over 1000 pages.

3.6. Flexibility of IEDMs

Exchange data models, especially the ones with a larger scope and many parties involved, tend to be never complete and finished. The modelling process takes too long for that. Also, the environment constantly changes, adjusting information exchange requirements as well. Regular upgrades of the IEDM are inevitable. When an IEDM is being released, this should happen with the knowledge it is not completely perfect and stable. One must take into account that the IEDM will change periodically, because it is being 'maintained' (corrected, extended) continuously.

One can cope with this fact by making the model as *flexible* as possible. This makes the model relatively easy to change and extend. Changeability actually means that corrections to existing semantics require little or no structural changes to the data model. Extendibility implies that new information types can be added without (or with minimally) changing the existing structure. This also ensures *backward compatibility*, the ability of systems to keep working with old versions of the standard.

Changeability and extendibility can be achieved by specifying the basic information elements in a data model in a *generic* manner (up to a certain extent, see par. 4.2). Because generic data structures describe reality in a universal way, modifications not to drastic do often fit in the model as it is (except for additional domain values).

Besides flexibility, proper change and version management is required to deal with periodical changes.

JC3IEDM is quite flexible, apart from several fairly detailed parts (see par. 3.4.3).

3.7. Naming convention for IEMDs

A proper set of naming rules is vital for an explicable and maintainable IEDM. The general aim is to have names that are readable and understandable, and as short as possible.

A well-working naming standard could look like this:

- entities and attributes get a single name, relations one or two names (inverse direction is optional);
- a name consists of one or more terms (words), separated by hyphens;
- entity names are written in upper case, attributes and relation names in lower case;
- entity and attribute names are nouns, relation names are verb phrases;
- entity names are (only) repeated in the names of (alternate) key attributes or subtype discriminators;
- attribute names end with a 'class word' (see below).

A naming convention could be much more detailed than this; these rules embody the most essential part.

A 'class word' indicates the nature of an attribute. It (partly) determines its meaning, usage and datatype. The use of class words in a data model proves to be quite valuable, among other things because it makes attributes more consistent and more comprehensible and because it obliges the modeller to consider carefully of what kind some attribute is. A useful set of class words is the following:

Class word	Purpose
identifier (id)	A meaningless integer[4] that uniquely identifies an object instance. It is *the* key of an *independent* entity. Example: person-id, i.e. an arbitrary integer to identify a specific person.
index	A meaningless integer[4] that points out an object instance, together with identifiers and (possibly) other indexes. It is *part* of the key of a *dependent* entity. Example: person-status-index, together with person-id the key of PERSON-STATUS.
code	An (often meaningful) string of characters that represents a specific qualitative property of an object out of a *limited* set of possible values. Most codes are abbreviations. A code attribute may be the primary key of a static entity (domain). Examples: unit-status-code (OPER, RECOV, MOV), nation-code (NL, NO, PO), month-code (JAN, FEB, MAR), married-indicator-code (Y, N).
number	An (often meaningful) integer or string of characters that designates an object or a property of an object in a sequential way, using numerical and/or alphabetical ordering. A number may be an alternate key. Numbers do not denote a physical quantity. Examples: house-number (5, 7A, 7B, 9), wind-force-number (0-12), document-annex-number (A, B, C), activity-sequence-number (1, 2, 3).
name	A word or phrase that designates an object or a property of an object. In the first role it may be an alternate key. Examples: PERSON.person-name, PERSON.employer-name, month-name.
description	A string of characters, usually in some kind of structured format, that denotes a property of an object, but not a name. Often, description attributes contain several information elements. Examples: person-address-description, zip-code-description (2536VA), licence-plate-description (QF-CU-73), NATO-stock-number-description, flight-number-description (KL601).
position	A string of characters containing the coordinates of a specific spot (on the earths surface, in airspace, on a map, in a building, etc.). Formats can be Latitude/Longitude, UTM, etc. A position is a special kind of description. Examples: unit-position (44.3N/05.8E), office-room-position (H-238 = building H, floor 2, room 38).
time	A string of characters denoting a specific chronological point in time, measured according to the Julian calendar. A time attribute may contain a date and/or a time. Any format is possible. A time is a special kind of description. Examples: activity-start-time (2004-07-24/14:00), version-date-time (16-08-2004), birth-year-time (1966), birthday-date-time (27-11), shops-opening-time (09:00).
text	An unformatted string of characters that describes a property of an object, generally in the form of words and sentences. A text does not contain any structured information elements. The length of a text attribute is limited (e.g. 256 characters). Example: commanders-intent-summary-text.
reference	A string of characters that are a link to a collection of unstructured data (documents, pictures, etc.) outside a data model. The unstructured data in total represents a certain (complex) property of an object, for instance a set of photographs showing the surroundings of a compound. The link can be a filename, URL, etc. Example: mandate-reference ("NATO-docs/mandate.doc").
count	A unitless integer that indicates the amount of some countable property. Examples: person-age-count, month-days-count, stored-articles-count.
fraction	A unitless natural or real number that indicates the valid proportion of some property. It has a fixed unit of measurement. Examples: probability-fraction (0.1), fail-fraction (50%).
quantity	A natural or real number that denotes a physical property of an object. A quantity attribute has a fixed unit of measurement (e.g. km/h). Examples: travel-distance-quantity, maximum-speed-quantity, activity-duration-quantity, load-weight-quantity, current-temperature-quantity.
angle	A natural of real number that indicates the rotational angle between two objects, assuming they can be represented as two lines/planes diverging from a common point/line. An angle has a fixed unit of measurement. Example: bearing-angle.

JC3IEDM has an excellent and extensive naming convention [4]. The only weakness in our view is that all attribute names include the entity name. This adds a lot of unnecessary text to the model, decreasing its readability.[5] Besides this, JC3IEDM uses a (slightly) different set of class words.

[4] Strings are possible, but undesirable because of key management (see par. 4.9).
[5] In the physical schema, however, denormalisation may require to do this for certain attributes.

4. Structural Guidelines for IEDMs

4.1. Introduction

This chapter contains guidelines about how certain concepts should be modelled in order to get the most effective IEDM.

4.2. Explicit versus generic

One can choose to model information in a more explicit or a more generic fashion. When explicitly modelled, real-world objects are (more or less) 'literally' incorporated in the data model. This means they are represented by specific entities, attributes and/or relations. Doing this has a number of benefits:

- the modelled objects are easy to recognise, which make the model more understandable in detail;
- objects require less constraints to be specified, because these are implicitly modelled;
- in general, it will be easier (more straightforward, with less code and making better performance) to implement an explicit data structure and to manipulate explicitly modelled data.

Generic data structures represent real-world objects in a general (non-specific) form. This means that such objects are modelled by entities, attributes and/or relations that cover other objects as well, i.e. that represent a more broad concept (exposed by more general names). Generic modelling has the following advantages:

- less entities, attributes and/or relations are needed to represent certain concepts;
- generic structures are 'reusable' sooner and could be applicable for a broader range of functional areas;
- generic structures are more flexible and stable, i.e. changes and extensions to the model can be made relatively easy and will have less effect on the rest of the model (see also par 3.6);
- modelling in a generic way is usually less 'subjective' (separate modellers will come up with more comparable results).

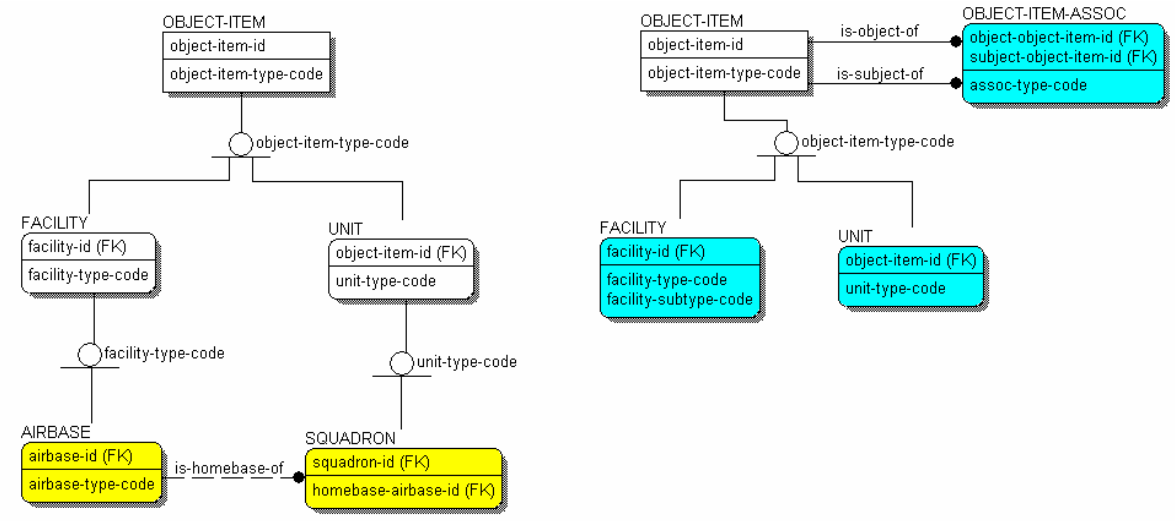


Figure 3 — Explicit (left, yellow) versus generic (right, blue) data structure

Figure 3 shows an example. In the left model the Airforce concepts AIRBASE and SQUADRON have been modelled explicitly, including their relation (which says that an airbase can host multiple squadrons and that a squadron has one home base). The right model illustrates how the same concept can be represented more generically. AIRBASE and SQUADRON are included in FACILITY and UNIT. Any

specific attributes are generalised (e.g. airbase-type-code into facility-subtype-code). The relation between AIRBASE and SQUADRON is represented by OBJECT-ITEM-ASSOCiation, hereby requiring an explicit constraint stating that a certain object-item of type 'unit/squadron' can only be linked to one object-item of type 'facility/airbase'.

The obvious question now is: how do I choose between these modelling approaches? The complexity of a data model is not really affected by the choice for generic or explicit structures. In the first case the model becomes smaller, but requires more additional constraints; in the second case the model is bigger, but needs less constraints.

Generic modelling origins from the notion that representing *every* object (in a certain context) as a *separate* entity will result in an unworkable model. In order to build systems, one has to abstract (= simplify) the real world into a model. Since many objects will have (almost) equal properties, they can therefore be united into the same entity. For example, the Airforce will probably require to distinguish between helicopters and aircraft, and within the latter class between transporters and fighters. But a distinction between F-16 and F-28 fighters may be less useful, because they can be described by means of similar properties (e.g. the use of the afterburner).

The main advantage of generic modelling is that it results in flexible and widely applicable models. A relatively small model is able to cover many concepts. During the modelling process, when new concepts are being added to a generic data model, the model often does not have to be changed very much.

Especially for dedicated and detailed concepts generalisation is beneficial. Such concepts tend to be quite unstable in the sense that they are sensitive to modelling variations and model changes. This can be coped with by generalising them. For example, military people may have numerous kinds of qualifications, acquired via regular education, courses, symposiums, training programmes, field exercises, etc. One could model all these kinds of tutoring separately (explicitly), because they have a different nature and hence seem to have some different properties. However, any other kind of education (initially not thought of) may appear to be just somewhat different in the details. Therefore it seems better to generalise this concept and make it feasible for all types of education. This could be done by means of an entity PERSON-EDUCATION with attributes like education-type-code, finished-date-time and passed-indicator-code.

Although this reasoning suggests generic modelling is very efficient, this is not really true. Firstly, it causes that certain fundamental real-world objects will not be visible in the model, making it harder to comprehend and implement. Entities such as AIRCRAFT and AIRBASE should occur explicitly in an Air C2 data model, because they represent vital objects that are used in many air-related processes. Such a model is much easier to understand than one in which these concepts are hidden in entities like MATERIEL and FACILITY. Manipulating explicit data inside a system will also be easier.

Secondly, a highly generic data model requires many extra explicit constraints. Also the number of attribute domain values increases. The loss of structure (less entities and relations) is 'compensated' by lager and more complex attribute domains. This means that generic models are not that flexible and broadly usable as it looks like. Reusing entities to represent new concepts (e.g. a ship is just another kind of VEHICLE) implies that several attribute domains have to be extended (e.g. vehicle-type with 'vessel', 'submarine', etc.). In addition, extra constraints have to be defined on the domains (e.g. if vehicle-type = 'vessel' then vehicle-subtype = 'frigate' or 'aircraft carrier' or '...', but *not* 'tank' or 'fighter' or '...'). Large and complex domains are obviously harder to define, maintain and implement.

Notice that the considerations above apply to the logical schema. The physical schema could still be made more generic, by denormalsation (see par. 4.3). Sometimes generic structures enhance the performance.

The guideline is as follows. Concepts that are very *common* in or *fundamental* to a wide context should be modelled in an *explicit* manner. This makes a data model more accessible, by ensuring these concepts are recognisable to users and easy to use in implementation. This is efficient since they are significant aspects

of the model and likely to be used frequently. Examples (in Air C2 context): AIRCRAFT, AIRBASE, SQUADRON.

Less basic or more *specific* concepts should be modelled as *generically* as possible. This is not only to limit the size of a data model, but also to make it more flexible and stable. Examples: model 'pilot' as a PERSON (no specific properties, no real need to represent explicitly); model 'squadron-crewchief' as an ASSOCiation between UNIT and PERSON; model 'aircraft-readiness' as an object-item-status. Also, do not try to model every exception. A model should in the first place reflect the key issues of the IERs. Certain specific and rare concepts may indeed require an exclusive data structure (e.g. an extra subtype), but it is often better to model such concepts generically.

In conclusion, the choice between explicit and generic modelling is difficult and requires careful consideration. At least, one should strive for a model in which the level of explicitness is not too diverse. A model with both very generic and very explicit structures may be unbalanced, making it illogical and more difficult to use.

JC3IEDM is not very balanced in the sense of generic and explicit structures. Some parts are very generic, sometimes too much, while other parts are rather explicit. Some examples:

- The object items PERSON and MATERIEL are very generic, while FACILITY and FEATURE are much more explicitly modelled with several subtypes.
- MASS-GRAVE seems to be a concept that is too explicit for this model. It could have been represented as a FACILITY.
- The ACTION structure of the JC3IEDM may be too generic. The main characteristics of some action are described by its objectives and resources. These have been modelled as sets of object items and object types. This is a bit abstract. For example, one can hardly specify *how* the resources are used.
- The relationship between different object items is correctly modelled at the level of OBJECT-ITEM, by OBJECT-ITEM-ASSOCIATION. Older versions of the model (LC2IEDM) used to contain (almost identical) explicit ASSOCIATION entities for every relevant link between two of the five OBJECT-ITEM subtypes. The current, more generic solution is more appropriate.

Considering JC3IEDM in total, it is rather generic. MIP argues that one of the benefits of this is that most general Air and Maritime concepts will easily fit in this former Land-based model. Only some new domain values have to be included. However, this will undoubtedly become a problem. Not only due to the disadvantages of a generic model already mentioned, but also because Airforce and Navy users will not recognise their specific data, making the acceptance of JC3IEDM as joint model hard.

4.3. (De)normalisation

Normalisation is a method by which relational modelling should be performed. It avoids duplication of data and potential inconsistencies because of that (see literature for further explanation). In a fully normalised data model all existing relations between data elements (attributes) have been explicitly modelled, by adding extra entities and relations and by using the right keys. Normalisation is rather similar to explicit modelling, but with different goals.

In general, normalising data models is a good practice. But normalisation also has a serious drawback. When certain complex objects or phenomena out of reality are modelled exactly as they are and in a normalised manner, this may involve a lot of entities and relations. Theoretically such a complex structure may represent reality correctly, but practically it will be hard to comprehend and utilise.

Normalisation also introduces many entities which are 'artificial' in that they do not really represent an existing object anymore. For instance, JC3IEDM contains the entities ORGANISATION-ACTION-TASK-RULE-OF-ENGAGEMENT-STATUS and OBJECT-TYPE-ESTABLISHMENT-OBJECT-TYPE-DETAIL; hardly anybody who's not a MIP modeller will be able to imagine what these stand for.

Therefore, sometimes it is better not to fully normalise, i.e. to 'denormalise'. Although it brings extra constraints, these are usually rather straightforward. Exact rules by which entities should or should not be fully normalised, are hard to give. A guideline to follow is to check whether entities are still recognisable. Too abstract entity names often indicate too much normalisation. For subtypes it is helpful to check whether their attributes, domains and/or relations with other entities differ considerably. If not, don't use subtypes. Instead, put the attributes in (and link the relations to) the supertype entity and add the necessary constraints to the entity.

JC3IEDM contains some structures which might have been denormalised:

- The LOCATION part of JC3IEDM contains 22 (!) entities. Its main intention, to describe all kinds of 0 to 3 dimensional geometrical structures, can in principle also be reached by just 2 entities: LOCATION and LOCATION-POINT, describing a location as a set of points making up a certain kind of shape (see figure 4). This is an (extreme) form of denormalisation, but would result in a structure far simpler.

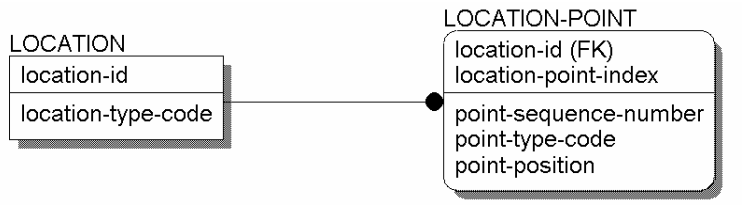


Figure 4 — A simple location structure

- The subtypes of OBJECT-ITEM-STATUS contain quite similar sets of attributes. There are of course differences (also with regard to the attribute domains), which endorses normalisation, but with a few extra constraints these subtypes may not have been necessary. The data structure would have been equally clear, but less complex.
- The subtypes of AFFILIATION contain just one or two attributes each. Simply putting them inside AFFILIATION itself would not much decrease semantic expressiveness, but it takes out 5 subtype entities.
- Finally, the next subtype tree shows how big a model could become when fully normalised. To find a unit type one has to go through 5 subtypes: OBJECT-TYPE → ORGANISATION-TYPE → GOVERNMENT-ORGANISATION-TYPE → MILITARY-ORGANISATION-TYPE → UNIT-TYPE. The limited number of attributes in some of the intermediate entities reveals that they are not really necessary.

The reasoning above applies to the logical schema of a data model. In a physical schema denormalisation may (should) be applied on a wider scale. This will enhance the performance of the exchange language. Having less entities implies less relations and foreign keys, which decreases the number of joins. In contrast, notice that other forms of physical optimisation may imply the opposite. Sometimes it is useful to split up tables (either horizontally or vertically), which increases the number of entities, but still results in better performance. Such optimisations often depend on how the data is used in practice.

4.4. Constraints

The schematic features of a data model enable one to define the basic structural rules of real-world objects. It does so by means of entities, attributes and relations. In addition, attribute domains specify which object property values are allowed. Such constraints are *implicitly* enforced by the data model. Other, more complicated restriction have to be defined *explicitly*, outside the model's schema, in plain text

or using some formal description method. These explicit constraints denote all kinds of limitations to instances and attribute values, such as possible combinations of values in two or more attributes or the obligatory existence of instances in certain cases.

For simple constraints there is the option to model them implicitly or explicitly. This is very much related to the choice between explicit and generic modelling (see par. 4.2). For example, in a personnel data model the attribute absence-reason-code in EMPLOYEE cannot be 'pregnancy' for male employees. This can be modelled in two ways: either implicitly by adding two subtypes, MALE-EMPLOYEE and FEMALE-EMPLOYEE, both with attributes absence-reason-code that have (slightly) different domains, or by defining an explicit constraint like "if PERSON.gender-code = 'male' then PERSON.absence-reason-code ≠ 'pregnancy' ".

Explicit constraints have two disadvantages:

- People (users, implementers) tend to be focussed at the schematic part of a data model. The documentation that goes along with it (including the description of the explicit constraints), is often not read very well.
- When generating a database from a data model, only the implicit constraints are automatically converted into database rules (triggers, etc.).

The first difficulty is a more broad one and applies to the explanation of the model in general (what it means, how to use it, etc.). This could be solved by producing better (smaller, more readable, better structured, etc.) documentation as part of a data model. The second problem may be resolved by specifying explicit constraints in a more formal manner.

The advantage of using explicit constraints is that it often results in a less complex schema with less entities, attributes and relations. However, this is not always the case and depends on the kind of constraint. In the previous example about employees the second option is preferred, because one relatively simple explicit constraint makes the data model schema much simpler. But if there would be a lot more restrictions with respect to the differences in properties of employed man and women, then subtypes could be the best solution.

In conclusion, the main guideline here is not to be afraid to model constraints explicitly. They will exist anyway, because a data model is simply not capable of representing the more complex restrictions that apply to the information (especially when attributes out of multiple entities are involved). Strict formalisation of explicit constraints (for instance in an ontology, see also par. 3.2.2) may be useful, but is also difficult and time-consuming. The cost-effectiveness is doubtful.

JC3IEDM has an extensive set of constraints (over 50 pages [4]).

4.5. Metadata

An IEDM describes the information that is part of certain IERs. We call this 'normal' data, in military sense also 'operational' data. *Metadata*, on the other hand, is data *about* (normal) data. There are two types of metadata:

- about the definition of the data;
- about the contents of the data.

The first type is in fact the data model itself and includes entity names, attribute definitions, domain specifications, etc. Such metadata could in fact be modelled as well by means of a so-called metamodel. This will contain entities like ENTITY, ATTRIBUTE and RELATION. Such meta-entities should of course not be explicitly part of the IEDM, simply because the metadata is already implicitly present (the models structure).

Yet, there are IEDMs that incorporate some metadata of this kind. To obtain maximum flexibility they contain entities, named like OBJECT-PROPERTY, which make it possible to dynamically define

attributes. So, not only its value, but also the attribute itself (its name, datatype, definition, etc.) is included in the IEDM. New attributes, i.e. properties of objects, can be added 'on-the-fly'. We think this is a bad way of data modelling. The price to pay is an extremely generic model with all its downsides (see par. 4.2). In addition, the flexibility is not real. Decision support applications that process this data still need to understand the semantics and have to be changed anyway when properties (attributes) are modified or appended.

The second type of metadata specifies dynamic properties of the data with regard to:

- creation (e.g. creator/owner, creation date);
- distribution (e.g. replication nodes and subscriptions);
- modification (e.g. date of last mutation, history of mutations);
- security (e.g. classification level);
- quality (e.g. accuracy, reliability).

Normal data can be related to these kinds of metadata. For instance, ownership rules of distributed data require to associate each instance (data record) of the model to an owner (creator), being the officer or organisation who is responsible for that piece of information. This could be done by adding an owner-id attribute to every entity in the model.

However, metadata should not occur inside (the logical schema of) an IEDM. It 'pollutes' the normal data described by the model, which is derived from the IERs, stating *which* information is needed to be disseminated among a user group. Metadata comes from different requirements, concerned with *how* the information must be shared. This has more to do with the exchange *method* than the exchange model. So this metadata should be part of the standardised exchange mechanism. Putting metadata inside an IEDM makes it needlessly more complex.

But how to deal with such metadata then? In essence, metadata which is relevant for interoperability (and does not just fulfil a national/local need) should be standardised as well. There are two ways to do this. Firstly, metadata could be added to the *physical* schema of the IEDM. Its special nature should be clearly indicated then. Example: all entities get an attribute 'classification-level-code' in order to designate whether (and to what extent) pieces of information are classified or not. In JC3IEDM the owner of records (owner-id) has correctly been added in the physical schema. Secondly, this metadata is defined by means of a separate 'exchange metadata model'. MIP uses a replication model that standardises nodes, contracts, etc.

Anyhow, metadata should be defined outside the logical schema of an IEDM. The difficulty often is, though, to determine whether certain information is normal or meta. What is normally considered to be metadata can in certain circumstances be normal data. Such data should be part of the (logical) IEDM indeed. Some examples:

- Data distribution aspects may be an essential part of an IEDM that supports the functional area of Communication and Information Systems (CIS). So, information types with regard to nodes and subscriptions will probably be contained in such a model.
- The classification level of a person (not: of a PERSON record) is rather normal data than metadata.
- Constructs similar to the one for ownership (see above) may be used to express that certain subsets of information are related, for instance because they apply to the same operation or show a situation from a specific point of view (e.g. the G3 overlay in a COP). JC3IEDM rightly uses CONTEXT for that.
- It may be useful to record metadata about the conditions under which facts (data) have been observed and reported (who, when, how reliable, etc.). Normally, this is genuine metadata. In JC3IEDM it seems that the entity REPORTING-DATA serves this purpose. It is used to keep record of where any piece of dynamic data came from (and also to enable versions of data, see par. 4.6). In this sense the reporting information is rather metadata than normal data and should be kept outside the model.

However, it could happen that the concept of the report itself — i.e. the actual message (SITREP, INTSUM, etc.) — is explicitly part of the IERs. In that case a report is treated as normal data, equally as objects like 'order' and 'mandate'. Apparently one requires to identify these information carriers explicitly, apart from the information contents inside. This is not the case in JC3IEDM, but still a few elements of REPORTING-DATA can actually be regarded as normal data, for instance the valid time of a unit location.

We recognise that the boundary between pure metadata and 'semi-metadata' is not always clear. Further discussion on this matter may help.

4.6. Versions of information

One can represent a real-life object 'as it is', but it may also be useful to record multiple *versions* of that object. Usually, this means the object changes over time and the historical properties have to be captured for some reason. The status of the object depends on time, which indicates when a certain status was valid. Besides time, there may be other dimensions that cause versions of objects, for instance different reporters (e.g. that observe the location of some tank differently).

How to model data that depends on time (or other version dimensions) is still not completely solved. Experts on relational data models are struggling with that for years already. Thus we do not have the ambition to solve this ourselves. On the other hand, a few simple guidelines can be given to deal with such information in a consistent way.

To start with, only add time where really needed. Most historical data will never be used in practice. To track the movement of a unit seems useful, so locations from the past should be kept. But, like JC3IEDM does, to record the history of object item possessions (HOLDING) seems already a bit questionable, not to mention the type an object item is (OBJECT-ITEM-TYPE, see also par. 4.7).

Only use versions when it does *not* involve metadata. This should not occur in a (logical schema of an) IEDM in the first place (see par. 4.5). For example, keeping record of mutations on data is a meta-issue.

In entities with versions of data, do not put the version attribute (e.g. time) in the key (see par. 4.9). Use an index attribute instead.

In the physical schema of an IEDM, it may be helpful to put current and historical data in *separate* entities. This will highly increase the performance when implemented, because current information is much more used in practice. For example, put the current location of an object item inside OBJECT-ITEM. This is redundant, because OBJECT-ITEM-LOCATION will contain it as well, but much faster when that data is retrieved. Redundancy is a form of denormalisation.

Versions in JC3IEDM depend on time and reporters simultaneously. It is realised with the REPORTING-DATA entity, which represents a 'report' of certain dynamic information (e.g. a new unit location), along with a number of qualitative attributes (source, reporting time, accuracy, etc.). All entities that embody highly dynamic data are related to this central entity (are 'loggable'), hereby recording the circumstances under which facts have been reported. REPORTING-DATA brings a lot of trouble. It adds much complexity to the model and introduces some serious performance problems, already when applying simple selections on a JC3IEDM database (e.g., select the *current* location of this unit). In case a construction like REPORTING-DATA is used in an IEDM nevertheless, then at least the same report record should *not* be re-used for multiple facts. This makes usage of the data even harder.

We think a better solution is the following (see figure 5). First of all, get rid of the metadata that REPORTING-DATA includes. If needed, the 'loggable' aspect must be solved at meta-level (see par. 4.5). Then put the remaining attributes that have to do with versions (e.g. 'time') *inside* the dynamic entities themselves. Now REPORTING-DATA may not be needed anymore. This highly simplifies the model as well as its usage.

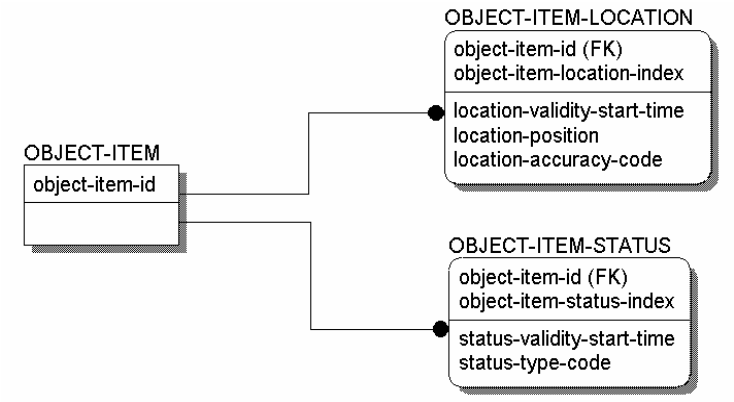


Figure 5 — Versions of object item location and status

4.7. Item versus type

There are two kinds of information one can register about real-life objects. The individual object itself can be described by means of its name, location, status, etc. This deals with a particular instantiation of the object, also called an *item*, which can be pointed out or even touched. An item refers to an object of which there's only one and which is distinctive from the rest by certain unique property values. Example: a specific passenger car with license plate "YA-04-RA", coloured blue and owned by Mr. X. Besides that, one can also describe a class or category or *type* of object, representing a group of items that have certain kinds of properties in common. Example: passenger cars, used to transport people and with certain maximum dimensions.

Items and types are always related: an item is of a certain type. For instance, MATERIEL points to MATERIEL-TYPE. For types, nesting is possible. MATERIEL-TYPE could refer to MATERIEL-CLASS.

An item must always refer to exactly one type, simply because that's the nature of items and types. In JC3IEDM this is not the case: OBJECT-ITEM-TYPE enables to link items to one or more types. Of course, it is imaginable that multiple perceptions exist about the type of some unidentified object, but in practice this will not happen very frequently, while it can be solved in other ways as well (temporarily using a type 'unknown object' or using multiple items of different type).

The distinction between items and types should always be made clear. The name of an object (the entity name) must clearly indicate whether it is an item or a type. For instance, ARTICLE is a confusing name, because it could both represent a type of material and an individual material item. In this case, ARTICLE-ITEM versus ARTICLE-TYPE is better. An obvious distinction between items and types is very important, because it will prevent modelling errors. But many people seem to have trouble with the difference. In our experience, talking with domain experts often requires to be continuously explicit about the nature of a certain object, for instance a 'tank': do they mean some tank on the battlefield (item) or the category of armed vehicles (type)? This will result in correct data structures, where an attribute like tank-colour is placed in the right entity (either TANK-ITEM or TANK-TYPE).

The use of types should be limited. This is because they mostly contain static data. The next paragraph about static data continues the discussion about types.

4.8. Static versus dynamic

Entities that represent objects of which the information (almost) never changes, are called static. They contain 'encyclopaedic' data. In most cases it concerns 'type' information (see previous paragraph), but 'non-type' static information does exist, for instance geographic features (nations, cities, rivers). Paragraph 3.4.4 suggested in principle not to include static data in an IEDM. For JC3IEDM this applies to

the constructions of OBJECT-TYPE, ESTABLISHMENT and CAPABILITY. Below, we briefly state some considerations on this matter.

4.8.1. Dynamic types

Is an extensive specification of types at all necessary? Since type data is usually static, it should not be part of an *exchange* data model. Information about the weight and dimensions of equipment is relevant, but also standard and therefore not required to be exchanged between systems on a regular basis. It is something that users need to have available all the time in order to understand the dynamic information they get. Still, quite a big chunk of the JC3IEDM contains type information, see the endless subtyping under OBJECT-TYPE. Although the bulk of encyclopaedic type data is only replicated once, at the beginning of an exercise, it should also have been defined outside the model.

This raises another question: is type information *always* static? Looking at the real world, this is common practice. Types are a static part of our speaking language; most words represent types (concepts). Everybody knows what a *car* is, so we can use that word without explaining it all the time. Introducing a new concept does not happen very often. This is like making up a new word on-the-fly. When speaking, we mostly introduce new items, for instance when you tell about your new car, an instantiation of the type 'car' with specific individual properties like brand and colour. Applying this to IEDMs means that type information should mostly be static.

On the other hand, there are particular situations where it does occur that types are dynamic. For instance, soldiers discover a new (previously unknown or not yet specified) type of weapon which is used by the enemy. This has to be added to the initial set of type data, as a new record in MATERIEL-TYPE. Still, this does not seem daily business and exchanging types for that reason is doubtful. A better example is this. Organisations that supply materiel (like depots) will frequently have to deal with new types of articles. New articles appear on the market, buyers require other kinds of articles, etc. In a Logistical IEDM the entity MATERIEL-TYPE will certainly be dynamic.

It may also happen that a type is dynamic just because one attribute is. For example, the entity EMPLOYEE-TYPE may contain a fixed number of employee roles with static descriptive attributes. However, the entity is still dynamic, because one of its attributes designates the maximum salary, which changes on a regular basis. The static attributes should be left out.

Thus, depending on its purpose, an IEDM may contain dynamic type entities. Whenever some data is considered dynamic depends on the scope and purpose of the model. For most domains a change frequency of less than a year can be regarded as static.

4.8.2. Entities versus domains

Although static data should rather not be part of an IEDM, it is still often necessary to refer to it. This brings us to another consideration: should certain static (type) data be explicitly part of the model at all (as separate entities) or should it be part of the domains? Since domains are static sets of data values, they could contain static data as well. This simplifies the model and avoids having separate static entities with a record that specify the possible instances. The entity PERSON-TYPE in JC3IEDM, for example, could also be modelled as a person-type-category-code attribute in PERSON. This is much simpler. Another example is the repeated use of nationality in different parts of an IEDM: model this either as a NATION entity to which other entities refer or use attributes nation-code (all with the same domain) in these entities.

It becomes somewhat harder with more complex static data structures, such as type entities that have subtypes. This is solved by using multiple attributes and constrained domains. For example, a part of the type tree in JC3IEDM is OBJECT-TYPE – FEATURE-TYPE – CONTROL-FEATURE-TYPE (see figure 6a). Instead of this structure, one could also utilise attributes. Two options are:

- Put attributes object-type-code, feature-type-code and control-feature-type-code in the entities OBJECT-ITEM, FEATURE and CONTROL-FEATURE respectively (6b). They also serve as discriminator attributes for the subtypes, making the OBJECT-ITEM discriminators superfluous.
- Suppose only OBJECT-ITEM is present in the model, as an (extreme) consequence of other guidelines. Put in there the attributes object-type-code, object-subtype-code and object-subsubtype-code (6c). Extra constraints are now in place to rule out certain combinations of values, for instance: if object-type-code = 'FEAT' then object-subtype-code = 'CTRL-FEAT' or 'GEO-FEAT'.

The general guideline for static type data is to make use of domains as much as possible.

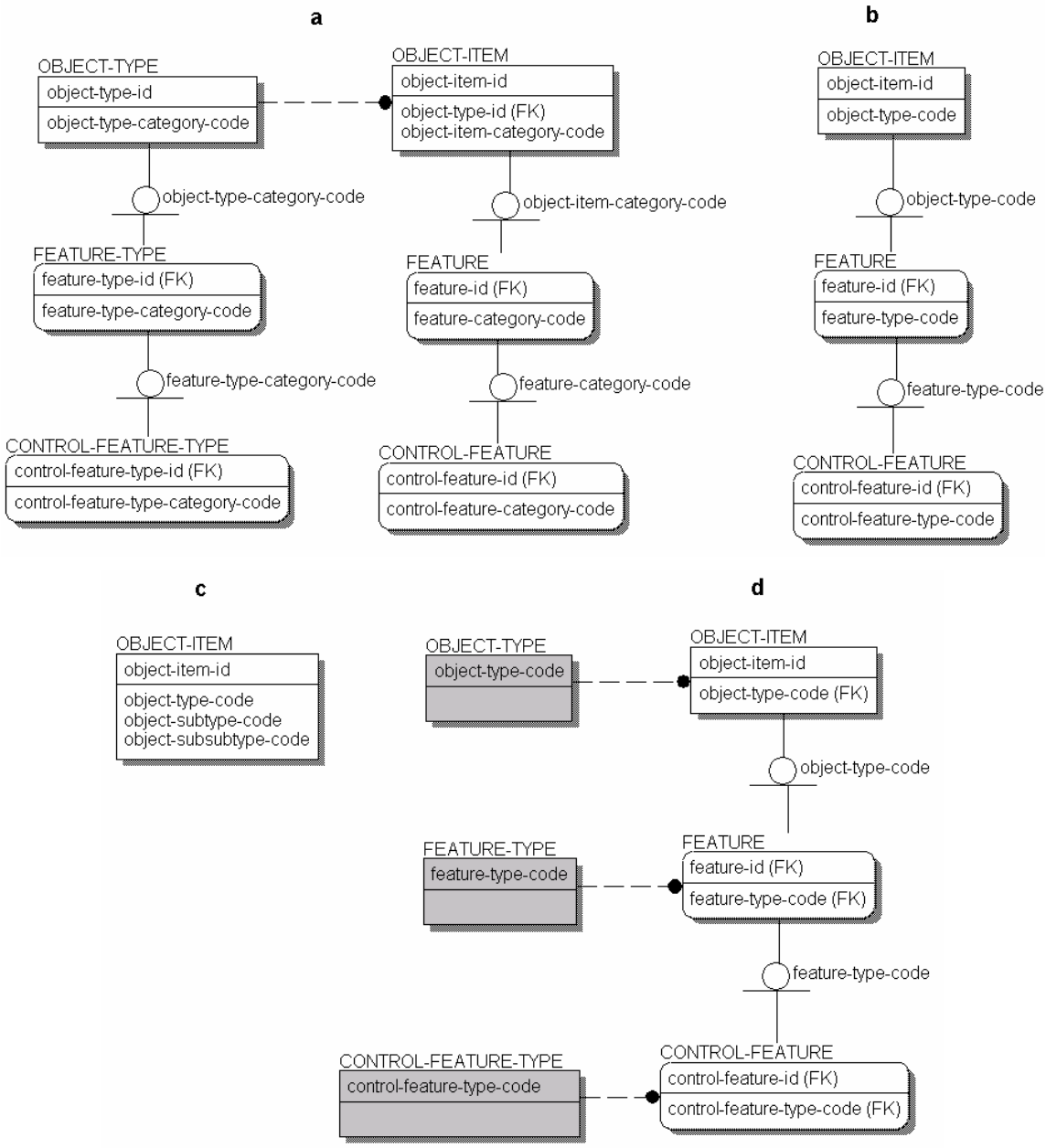


Figure 6 — Options for modelling complex types

However, for reasons of making the model complete and recognisable, it is sometimes better to make the types visible nevertheless. The solution here is to include the domain tables as entities in the data model, but clearly distinctive from the other entities (e.g. displayed in grey). Example: NATO Logistics uses five classes of materiel (I - V), which imply important directives for transport and storage. This little set of static data should be a domain, but because its operational relevance it could be made part of the model as the 'domain entity' MATERIEL-CLASS. In the previous example this principle is visualised as option 6d.

Normally domain entities just have one attribute, the key. This is always a 'code', not an 'id'. In some cases it may be useful to include additional attributes in the domain entity. These should be of high relevance to the rest of the model. For instance, properties like the 'arm' and 'size' of UNIT-TYPEs are very important (semantic) aspects of a unit type. They are also frequently used, among other things to plot units on a map by means of the right military symbols.

We admit that some of the guidelines about types and static data are somewhat contradictory. Further consideration of this topic is needed.

4.9. Keys

A little theory: An entity contains two kinds of attributes: 'primary key' attributes and 'non-key' attributes. The last ones describe the properties of a data instance (record). The primary key attribute(s) of an entity, together called the 'key', *identify* the instances. So, each instance has a unique combination of key attribute values. Relations between entities make that primary key attributes migrate from their 'base' entity to other (child) entities and become so-called 'foreign-key' attributes. A foreign key refers to an instance in another entity. Some non-key attributes can be alternatively used as key attributes to form an 'alternate key' in that way. Primary key attributes can never be NULL, foreign key attributes can. Below, a number of guidelines concerning keys are given.

Primary keys may never change, because they are being re-used throughout the model in order to relate instances to each other. Altering a key would imply altering many records. For example, if an object-item-id would change, then every data record related to that item need to be adjusted as well.

Keys may also not be controlled by external parties. This avoid problems when such a party suddenly decides to change the format of some id. For example, do not use the NATO stock number to identify MATERIEL instances, because that number is controlled by another body than the one responsible for managing the IEDM. The same applies to social security numbers for persons.

Key attributes should be *meaningless*. They may not contain any meaningful components, that contain some property or even refer to other attributes. For example, suppose the key of a piece of materiel contains codes that indicate the nation and organisation that owns it. Whenever that piece changes owner, these codes either have to be modified (which is forbidden) or become useless. When generated, a key attribute value may be composed out of such codes, as long as no one uses them afterwards. Meaningless implies only usable for identification.

A distinction should be made between primary key attributes from *independent* entities (called 'ids') and additional key attributes in *dependent* entities (called 'indexes'), see also the class word convention used in this paper (par. 3.7). An id (as primary key attribute) solely identifies the instances of some basic object in a data model. An index (as primary key attribute) is an additional designator for objects that are also identified by one or more foreign key attributes.

Ids and indexes are only used for *dynamic* data. Static data is identified using *codes*, the only other kind of key attribute. Example: the proportions of articles may be specified by length, width and/or height (depending on what kind of article it is). This can be modelled as done in figure 7, where DIMENSION-TYPE is a domain entity with values 'LE', 'WI' and 'HE'. A code can only be the primary key of a (static) domain entity.

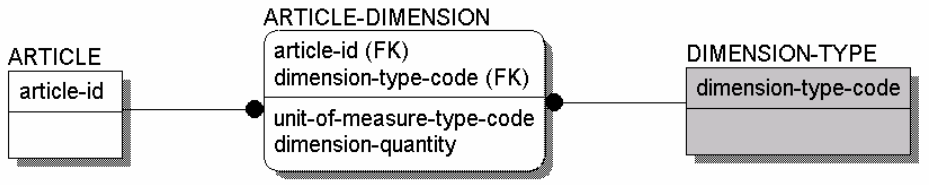


Figure 7 — Static domain entity with a 'code' attribute as key

Alternate keys can be name, number or time attributes. Sometimes, placeholders are needed for external or legacy ids. We use the 'description' class word for that.

Ids and indexes should be *integers* rather than strings of characters. This will ease the assignment of key values. Automatic generation of *new* key values and keeping record of already used ones, both necessary to keep keys unique, is very simple when using integers.

In a distributed environment preserving universal uniqueness of keys is not trivial. So, for an IEDM a *universal key management strategy* has to be defined. This prescribes, among other things, which set of key values is assigned to which party.

JC3IEDM uses keys in almost the exact way described here.

4.10. Independent entities

An independent entity is identified by a single primary key attribute which is not inherited from another entity (so is not a foreign key). The object an independent entity represents does not depend on any other object (within the data models context). For example, pieces of MATERIEL (independent) exist on their own, but data in MATERIEL-STATUS (dependent) can only exist when related a specific MATERIEL instances. Unlike IDEF1X, we consider subtypes of independent entities to be independent as well.

The general guideline is to limit the use of independent entities in a data model. This keeps the model simpler. Only the most fundamental, recognisable and concrete (non-abstract) real-world objects should be represented by such entities. This may depend on the models scope and context. For example, CONTRACT could be an elementary entity for an Enterprise Resource Planning (ERP) IEDM, but not for a C2 IEDM. In addition, these objects must really have an independent character. They exist on their own and can be fully described by themselves, independent from other objects. A separate list of LOCATIONs, like JC3IEDM encompasses, has no meaning.

Too often a data model contains independent entities that are rather properties of other entities than autonomous objects. They should be linked to these entities instead and become dependent. In JC3IEDM there are some unreal independent entities, for example AFFILIATION, CANDIDATE-TARGET-LIST, LOCATION and maybe CAPABILITY. An object item (or type) can have an affiliation (e.g. religion), being a property. The fact that an object item can have multiple affiliations is also not a rationale for an independent entity AFFILIATION. Figure 8 shows how this can be modelled. The same applies to locations: it would be better to repeatedly incorporate location attributes directly inside entities related to LOCATION, such as OBJECT-ITEM-LOCATION (see also figure 8). A reference to more complex geometrical structure can be modelled as one or more child or subtype entities under OBJECT-ITEM-LOCATION. LOCATION itself can be removed.

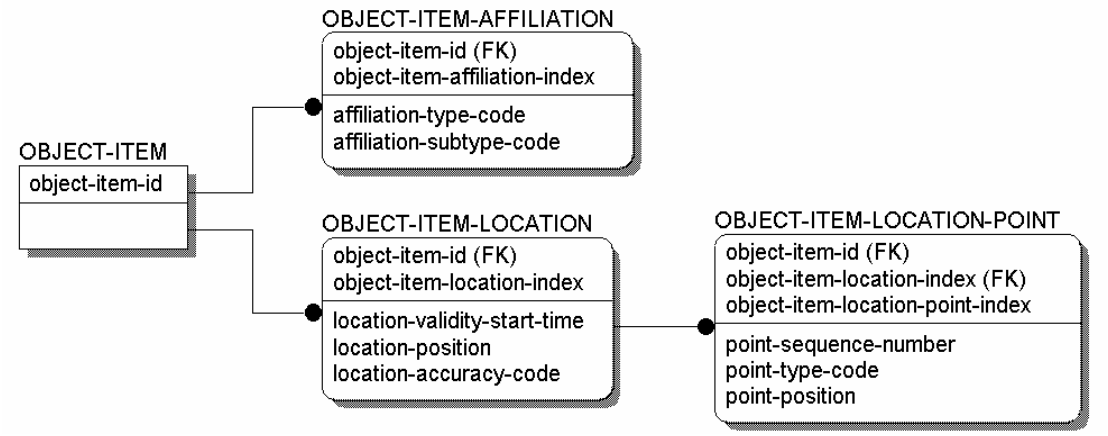


Figure 8 — Avoidance of independent entities

4.11. Unstructured information

Unstructured information can be fragments of free text, graphics, sound or video. These are usually quite large (amount of bytes) and can be found in documents and other kinds of files. Information should only be structured if there are good reasons to do so, for instance:

- many relationships exist between information elements;
- automated applications are available that perform certain manipulations or calculations on the information;
- there is a need to search and navigate a lot through the information;
- the amount of information is large and consists of many instances of the same type.

The common information exchange requirements determine the information which will be shared among a user group. In case the majority of this information is unstructured, an IEDM is of no use. A collection of documents (on a web site), for instance, would be more sufficient then. Otherwise an IEDM is needed to clearly and unambiguously define the (structured) information to be exchanged. However, it will nearly always be the case that a part of the information out of the IERs is still *un*structured. Examples in the Joint C3 domain are:

- NATO documents describing the operation's mandate;
- a presentation about the security situation in some area;
- a commander's intent and assessment;
- written reports on some incident;
- photographs showing the status of a bridge;
- UAV video fragments of spotted enemy troops.

This kind of unstructured information is very significant and also very much related to other *structured* information (such as units and locations). Therefore it would not be wise to separate the two. The combination of structured and unstructured information is considered increasingly valuably. To be more precise, we advise not to split structured and unstructured data with regard to the standardisation of *semantics* (IEDM). The actual *exchange* of structured and unstructured data could still be done along separate channels. Because unstructured data often involves much more space, one may decide to exchange that only on demand (using a pull mechanism), while the structured data is distributed on the basis of subscriptions (push or publish/subscribe mechanism).

Since a data model is meant to capture structured data, how can unstructured data be taken along then? The solution is to *link* the unstructured data to the structured data. The IEDM will incorporate attributes that are placeholders for (i.e. contain references to) documents, pictures, movies, etc. Also *sets* of these may be referenced, by specifying some directory. For instance, the entity ACTION-EVENT may contain

an attribute action-event-reference that is an optional link to a number of additional descriptions about some incident. FACILITY could have an attribute facility-view-reference that points to photographs of specific facilities. And passport-photo-reference is an obvious attribute in PERSON. By using these references, the (container of the) unstructured information has become part of the information exchange standard.

The unstructured information will be defined outside the models schema. One should standardise on the possible file types and formats. For example, pictures in JPEG, text in RTF, etc. Also the link itself should be standardised, probably by making use of URLs (Universal Resource Locators, i.e. browser addresses).

Limited portions of textual information can be put in regular text attributes instead of distinct files, making it easier to access. For instance, (a summary of) the commander's intent could be an attribute commander-intent-summary-text in the entity ORDER (not in JC3IEDM).

It may be useful to define a small set of (structured) metadata — outside the IEDM — about the references. Common properties such as reference type (document, picture, etc.), name, topic, source and date could be useful for navigating through the collection of unstructured information.

There is a lot of unstructured information relevant to be integrated with information from the JC3IEDM. The list above gives some examples. Especially the Consultation part of the model, supporting deliberations amongst authorities, implies the exchange of much documents.

Besides a few text fields, JC3IEDM has an (independent) entity called REFERENCE. It enables to link all reported information to a document. The major problem of this approach is that the unstructured information 'is not really modelled along with' the structured information. Any kind of reported data can be linked to additional documents, which is a very generic solution. Often it is possible to be much more explicit about what kind of unstructured information is expected to be associated with certain structured information (see examples above). Notice also that non-reported data in JC3IEDM cannot be related with unstructured information. A final shortcoming is that only one document per report can be specified.

5. Guidelines for Multiple IEDMs

5.1. Introduction

As we have seen in paragraph 3.4.2, proper scoping leads to the notion that information exchange in a certain context may require to define more than one IEDM. This chapter contains a summary of how this could work (par. 5.2) and what specific modelling guidelines are involved (par. 5.3 and 5.4).

5.2. Information Interoperability Domains

5.2.1. Introduction

The concept of "Information Interoperability Domains" forms the basis for multiple IEDMs. A previous paper [1] explains all about these domains. The paper promotes a simple 'common sense' idea: achieve *enterprise-wide* interoperability at semantic level by *splitting up* the 'information landscape' into relatively *autonomous* parts, which can be *independently* defined, but still *fit* in the overall plan of information exchange.

System interoperability could be achieved by using dedicated interfaces between each pair of heterogeneous systems that have to be connected (bilateral exchange). When more than a few systems need to be made interoperable, a common exchange language is the preferred solution. It results in a *minimum* number of interfaces and offers *maximum* independence and flexibility.

In support of this, the concept of 'domains' is introduced. The set of systems that interact by using the *same* exchange language is called an *information interoperability domain*. Each domain has its own standardised exchange language. A system is said to be part of a domain when it *is able to interact* with other systems by making use of the domains exchange language. Figure 9 shows how domains are drawn; notice that a 'domain' represents both a set of systems *and* an exchange language. The size or *scope* of a domain determines *how many and what kind of systems* belong to that domain. Since a domain represents an information standard, its scope can also be specified as the *kind of information* that is common and exchanged between systems within the domain. The scope must be clearly defined in order to avoid wrong usage of the standard.

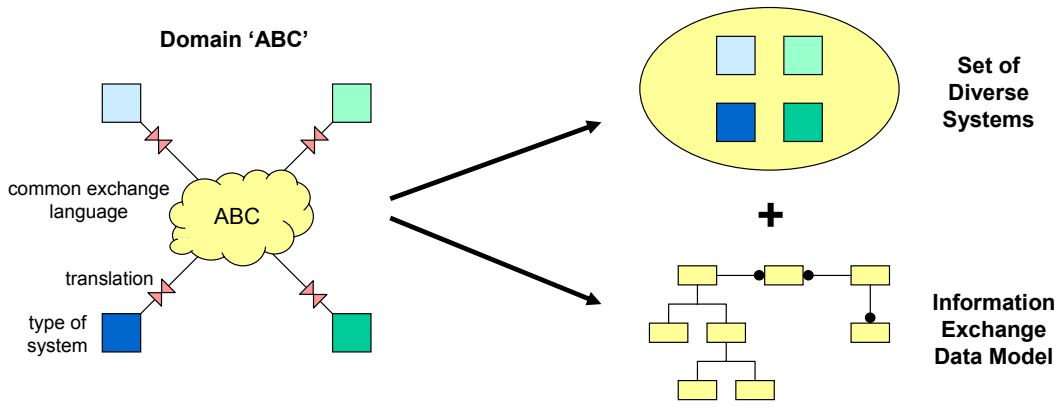


Figure 9 — Representation of an interoperability domain

5.2.2. The perfect (but impossible) solution

The approach of a common exchange language in itself is *not* feasible for large areas such as NATO C3. Theoretically, the ideal solution for enterprise-wide interoperability would be a *single* information standard ('Esperanto') that covers all information exchange between all systems. It would result in the

simplest (and cheapest and fastest) solution. But due to many reasons this solution is unlikely to be ever achieved. Key problems are the volume and complexity of such a standard and the implicit time to develop it and agree upon it. The broad context implies many parties involved in the standardisation process, each with its own policy, plans, standards, etc. This will make a consolidated exchange language difficult to obtain; the end result will always be incomplete and out of date. Therefore, a subdivision into *multiple* exchange languages — each with a specific (limited) scope — will be necessary. This is quite an extraordinary view on interoperability within NATO, especially with respect to MIP, because JC3IEDM seems to be meant to become this (utopian) all-enclosing IEDM.

Having multiple exchange languages implies having multiple interoperability domains. Systems can belong to more than one domain; in that case they 'talk' multiple languages.

5.2.3. Multiple domains

So, in case of a large environment, having *multiple* domains is usually unavoidable. The real challenge in this is finding an optimal partition of the total interoperability area. This comes down to finding a proper *scope* for each domain as well as the best *subdivision* of the whole area *into* domains (see par. 5.3).

A domain typically contains systems with much information in common, which is exchanged between them relatively frequently. So a domain represents a clustering of systems which are closely related[6]. This implies that a suitable initial set of domains is basically derived by grouping systems that exchange much similar information. But enterprise-wide interoperability, being a NATO requirement (see par. 1.1.2), implies there is a need to interconnect potentially any system with any other. This means that systems out of *different* domains must be able to talk with each other as well. In other words, domains have to be linked somehow. This happens in the same way individual systems are clustered into domains: the initial (or 'basic') domains may be grouped into new domains at a higher level. A higher-level domain represents a set of domains (groups of systems) which are interconnected by means of a common exchange language.

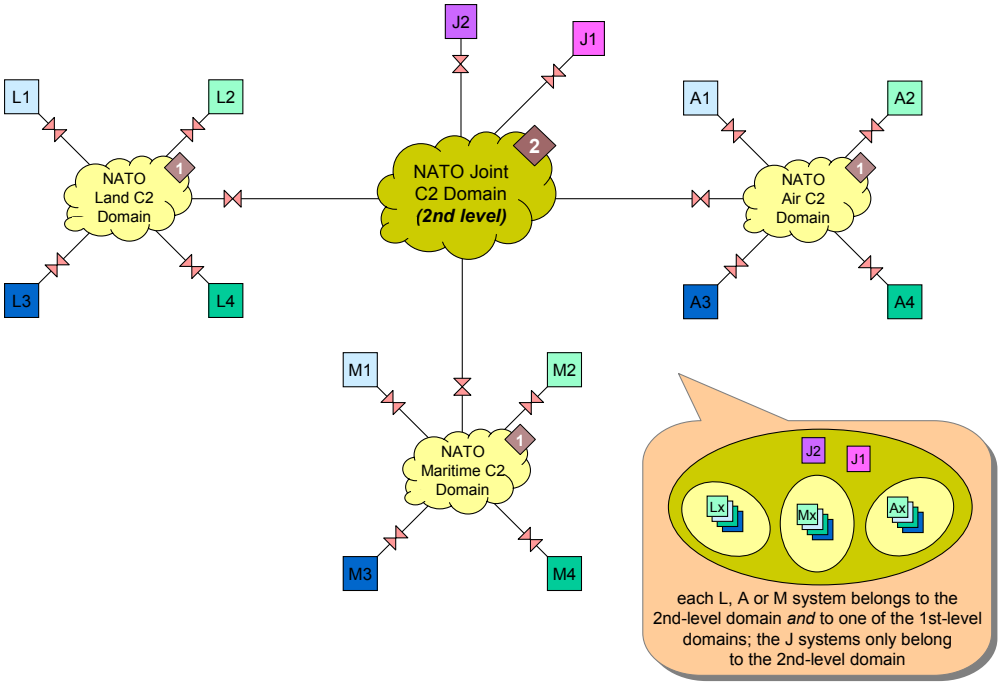


Figure 10 — Three domains with a higher-level domain in-between

[6] Besides similar contents of information (being the primary factor), the grouping of systems can also be affected by resemblance in the functionality of systems, quality requirements (such as security and timeliness) and organisations that use or own the systems.

For example (see figure 10), suppose we have three basic interoperability domains: the Land, Air and Maritime C2 domains of NATO. If systems of *different* domains want to interact they could use one of the three first-level languages, but this solution is only feasible when just a few domains exist. A better option is using an *additional* exchange language, through which systems of different domains can interact. In this way, a *new* interoperability *domain* has been created at a second level, acting as 'glue' between the three first-level domains. Besides to their basic domain, all systems also belong to this "NATO Joint C2" domain (as shown in the balloon). The advantage of this solution is that it requires less languages to be 'learned' by systems.

A higher-level domain may also act as 'basic' domain for certain systems. These systems use the exchange language of that domain to talk to each other. The figure shows two Joint C4I systems (J1, J2).

5.2.4. Domain structure

So, when the number of domains increases, the exchange between systems out of *different* domains can be standardised again, by defining information standards (domains) at *higher levels*. This results in an domain *structure*, which enables interoperability between a large number of heterogeneous systems. It is an efficient and flexible solution in that it limits the number of dedicated interfaces. It is also a realistic solution, because it avoids giant universal information standards.

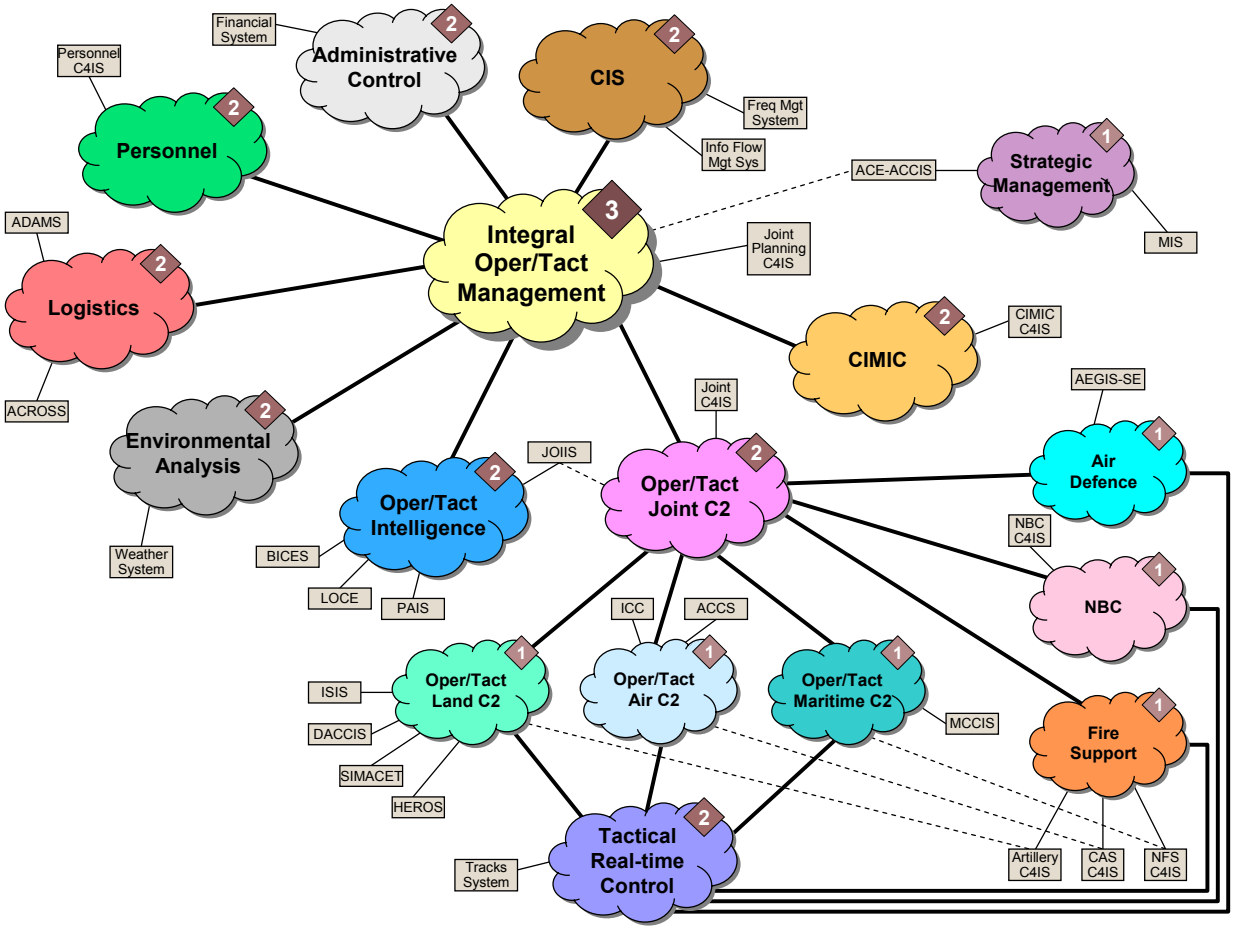


Figure 11 — A possible NATO C3 Domain Structure

Figure 11 shows a possible domain structure for NATO C3. This is just an example of how it could look like, based upon a brief information analysis [1]. You can see the NATO C3 information has been subdivided into 17 interoperability domains. They represent clusters of interacting systems, using their own IEDM. The various IEDMs are supposed to have limited overlap. Some *example* C4I systems have been added (as little boxes) for explanation purposes. The domains are connected in such a way that between each pair of systems interaction is possible. Information is exchanged by talking an 'in-between' exchange language.

A quick walk-through reveals that there are separate domains for basic Land, Air and Maritime C2 systems (at operational/tactical level), which are all included in a second-level domain "Joint C2". They also belong to another second-level domain, specifically for exchanging tactical real-time information like tracks. Certain very specialised C2 functional areas, such as Air Defence and NBC, also have their own domains and can also interact via the "Joint C2" and "Real-time" domains. Systems supporting other major functional areas besides basic C2, for example Logistics, Intelligence and CIMIC, are placed in distinct domains as well. Managing an operation in totality requires the integration of information coming from all these functional areas; this is achieved by means of the "Integral Operational/Tactical Management" domain in the middle. Finally, systems at strategic level, supporting among other things the consultation process, are part of the upper-right domain.

Each system has a basic interoperability domain (to which it is directly linked), but also belongs to other (higher-level) domains connected to that. For instance, a MIP-compatible C4I system (e.g. ISIS) is part of four domains: Land C2 (basic), Joint C2, Real-time and Integral Management. If systems part of the *same* domain want to exchange information, they make use of the domain's language. For example, BICES and LOCE talk the Intelligence language with each other. Systems of *different* domains that need to interact, take the language of the *lowest*-level domain they *both* belong to (= *highest*-level domain in the shortest *chain* of domains between them). For example, ICC and MCCIS use Joint C2 language (or the Real-time language in case of real-time information exchange); ICC and ADAMS use the Integral Management language. Some systems are *directly* linked to more than one domain. The Artillery system uses the Fire Support language to co-operate with the Close Air Support and Naval Fire Support systems, but also talks the Land C2 language when it interacts with Land-based C4I systems.

Finally, be aware that the lines in figure 11 do *not* express information flows. Also, the sequence of clouds between two systems does not represent some kind n-step translation. The picture basically shows an hierarchy of collections of systems, that indicates which exchange language to use for interaction between two arbitrary systems. The rule is: take the *lowest-level* domain they *both* belong to.

5.2.5. Comparison with existing concepts

We realise this approach may raise a lot of questions, especially in relation to ongoing interoperability efforts. This paragraph contains some additional explanation which may answer some of these questions.

- **Scope of higher-level and lower-level domains.** One may think: the Joint C2 data model contains the *union* of the underlying Land/Air/Maritime IEDMs, so why is it a separate domain if there is so much overlap with these other domains? However, the joint model rather resembles the *intersection* of the Land/Air/Maritime models. In other words, it contains the information being exchanged *between* dissimilar Forces. This information tends to be more aggregated and less detailed. In addition, the Joint C2 model may include Joint-specific information (e.g. regarding joint planning), part of the IERs between joint commanders. Information used for Land, Air or Maritime (single-service) operations is included in one of the Land/Air/Maritime C2 IEDMs. Besides some general stuff (units, facilities, etc.), the models will contain specific entities and attributes that describe details about, for example, tanks, helicopters and submarines. The size of the Joint model is supposed to be comparable to the Service models.

- **Domains versus the NATO Reference Model and the Generic Hub concept.** Isn't the above presented concept of a Joint C2 model with Land/Air/Maritime models underneath equal to the (former) NATO Reference Model with its (intended) View Models? No, there is a subtle, but important difference. NATO worked towards a single all-enclosing standard. The NATO Reference Model was meant to form the core of a big data model. The View Models should have been 'hooked up' to that core. This is not the case over here: the Land/Air/Maritime models will *not* be connected to the Joint C2 model. The idea of the domain structure above is to keep the IEDMs separate and hereby limit their size. Instead, a hierarchy is created, which relates the different IEDMs with each other on the basis of an optimal scoping (subdivision of the whole NATO C3 area). A similar reasoning applies to the Generic Hub (i.e. the heart) of JC3IEDM: the IEDMs on the right of figure 11 (Air Defence etc.), all related to the Joint C2 model, are likewise different from the functional area extensions around the Generic Hub (see par. 2.2). The Hub *would* be a good candidate for a standardisation model, though (see par. 5.4).
- **The Joint C2 domain versus the JC3IEDM.** The (imaginary) Joint C2 model is not equal to the JC3IEDM. Their scope differs considerably. The JC3IEDM is intended to cover (as much as possible) joint information as well as all Land, Air and Maritime C2 information, whereas the Joint C2 model in the example above only covers joint information (i.e. information required at joint C2 level).
- **Translations.** It is true that having multiple IEDMs implies more translations between data formats than needed with a single IEDM. This is a trade-off with less complex and more stable standards. And if the IEDMs are well-scoped and compatible with each other (see further), then the translations will be minimised and simplified.
- **Stove pipes.** The domain approach is the middle between fully centralised control (one big model) and total individual developments. This keeps it manageable. But it will also not lead to another set of stovepipes, or 'islands' of systems, not able to talk to other systems. The domain structure and the 'umbrella' principle (see par. 5.4) avoid this. And, due to central scoping, there will be a limited overlap between the IEDMs, hereby getting round the risk of ambiguous domains (indistinguishable to which a system belongs).

5.2.6. Conclusion

An information interoperability domain structure may be a good solution to manage NATO interoperability. In terms of this paper, this means *multiple* exchange languages will exist. This has some significant implications for these IEDMs. The next two paragraphs explain the most important ones.

5.3. Scoping in relation to other IEDMs

Equivalent to what is stated in paragraph 3.4.2, the *scope* of a single domain should be as large as possible, while the domain's information standard (exchange language) remains manageable with regard to approval and development. The difficulty to reach *agreement* on a common information standard with many players involved and the *complexity* of modelling, implementing and maintaining that standard, limit the scope of an interoperability domain.

Exactly which kind of systems should be in which domains (or: which information should be in which IEDMs), is another question to be answered. The optimal *subdivision* into separate domains is such that the domains become as autonomous as possible. A domain should have a strong internal cohesion, meaning that it contains systems with much information in common, which is exchanged between them relatively frequently. At the same time, this type of relation with *other* domains should be weak and well-defined. This will minimise the overlap and thus the co-ordination between developers of different domains, hereby decreasing implementation costs and possible inconsistencies of standards.

As said before, JC3IEDM is too big at the moment. A better scoping according to the interoperability domain approach, could entail the following actions:

- Split JC3IEDM into a Land C2 model and a generic standardisation model ('umbrella', see par. 5.4). In fact, this is a reversal of the recent merge between C2IEDM and NATO Reference Model [2].
- Develop *separate* models for the functional areas of Air C2 and Maritime C2, both based upon that umbrella. Do so by creating distinct user groups. The models will contain the Air- and Sea-specific features, for example detailed descriptions of airbases and harbours respectively.
- Synchronise with other domains (such as Intel and Logistics) to refine the scope, possibly resulting in removing certain data structures (e.g. MEDICAL-FACILITY) from the Land C2 model.
- Set up a central board (e.g. under NDAG) that co-ordinates all modelling work with respect to interoperability in NATO. The board is responsible for scoping the domains (IEDMs), defining modelling guidelines and developing the umbrella model. Providing the guidelines are followed and the umbrella is used properly, the board will *not* interfere in the modelling activities for the individual IEDMs and does not approve the contents of these models.

5.4. Compatibility with other IEDMs

Inherent to a domain structure, systems will have to talk *multiple* exchange languages and *translate* them to and from their internal data structure. Therefore, the different IEDMs should be *compatible* to a certain degree, which makes conversion easier.

This could be achieved by letting all IEDM development teams (one per domain) apply the same modelling guidelines. Assuming this is a complete and proper set of guidelines (for which this paper is a start), it will ensure modelling is done in a similar way, resulting in comparable data structures.

In addition, all IEDMs should be based upon a standardisation data model (see par. 3.3.1). This is a common glossary containing basic information types and structures. By 'common' is meant: throughout the whole interoperability area, over all domains. The glossary standardises the most fundamental terms and concepts used in the IEDMs. Mostly, such concepts are incorporated in more than one IEDM. Using exactly the same data structures in that case, will highly improve compatibility between IEDMs. The model can be called a *'semantic umbrella'* in that respect.

In NATO C3 context, the semantic umbrella will define the meaning and structure of the most fundamental C3 information types. This includes simple attribute domains, such as:

- time (e.g. Date-Time-Group format);
- location (e.g. UTM);

as well as complete data structures, such as:

- basic object items and types, i.e. organisation/unit, materiel, person, facility and feature;
- a generic way to describe associations between object items;
- plans and events (actions).

Linked to each other, the data structures form a generic core structure or 'framework', a (small) data model with very common entities, attributes and relations. The core should be very generic, making it more stable. Changes in the core affect all models built upon it and must be avoided as much as possible.

Notice that this guideline does *not* imply the creation of one central data model with 'subfunctional appendices' (like the NATO Reference Model). Instead, we plead for the more or less independent development of IEDMs for separate information sub-areas. The interrelations between these areas are covered by the central core which all models must incorporate.

Each IEDM should be built around this core, possibly leaving out the things not needed. Domain-specific entities, attributes and relations are attached to the central core. Generally, the core will be small compared to the specific part of the model.

6. Conclusions

6.1. General conclusions

A common exchange language is a necessary instrument to achieve system interoperability. But making a high-quality IEDM is difficult. Such a model is often too large and complicated, while it has a very wide appliance. The guidelines in this paper help to make better IEDMs.

Interoperability is a hot issue in large organisations such as NATO. In general, they could gain a lot when they would pay more attention to modelling *practices*. This both includes technical issues about how to model a proper IEDM and more broad concepts such as the use of an interoperability domain structure. For NATO, discussions about pure modelling issues should probably be led by the MIP/NDAG community, because they appear to capture the most experience in this field. The overall strategy on system interoperability should most likely be debated at NATO policy level. In both cases this paper could be used as input.

The JC3IEDM embodies an excellent concept and is the best IEDM currently available within NATO. However, there are a lot of modelling aspects that could be improved, hereby considerably smoothing and thus increasing interoperability within that context. In addition, the scope of the model — Joint C3 — has become too big. And the argument that most general Air and Maritime concepts will easily fit in this former Land-based model (because it is very generic), will probably not hold. Airforce and Navy users will not recognise their specific data, making the acceptance of JC3IEDM as joint model hard. A solution to this could be the use of an interoperability domain structure. The moment MIP realises this and accepts the concept of *multiple* IEDMs, it takes a huge step forwards, on the way to broad usage of the MIP standard in practice.

6.2. Main guidelines for IEDMs

The most important guidelines for making effective IEDMs are the following.

General guidelines

1. A relational data model is a well-suited technique to specify an IEDM. The role of a data model, i.e. application, exchange (= IEDM) or standardisation, should always be clear. An IEDM should consist of three data model levels, conceptual, logical and physical, which must all be standardised. These levels make the data model design process easier and more transparent.
2. An IEDM must have a well-defined scope and purpose, which ensures the model will be used correctly. Furthermore, the scope of an IEDM must be limited such that the model remains manageable with respect to overall approval and development. The basic directive is to cluster systems that exchange much similar information and assign a single IEDM to them. Because these systems normally have comparable functionality, an IEDM should contain highly correlated information types.
3. An IEDM must be ‘balanced’ (not too much variation) with respect to the level of abstraction and detail. Static (type) information should not be part of an IEDM. Instead, it can be put it in a separate data model if needed, while references to types should be solved by using domains.
4. An IEDM must be as simple as possible, making it relatively easy to understand, develop and implement, otherwise it will never be applicable in a large environment. The complexity of a model is highly related to the number of entities, which can be reduced by limiting the scope and/or preventing full normalisation.
5. An IEDM should be as flexible as possible, making it relatively easy to change and extend.
6. A proper set of naming rules is vital for an explicable and maintainable IEDM. Especially the use of class words is quite valuable.

Structural guidelines

7. Concepts that are very common in or fundamental to a wide context should be modelled in an explicit manner. Less basic or more specific concepts should be modelled as generically as possible. The choice between explicit and generic modelling is difficult and requires careful consideration.
8. Although normalisation of IEDMs is a good practice, it is sometimes better *not* to fully normalise. Denormalised structures can simplify the logical schema. In the physical schema denormalisation should even be applied on a wider scale, hereby enhancing the performance of the exchange language.
9. It is often better to model complex constraints (business rules) explicitly, because this will reduce the complexity of the data model.
10. Metadata should not occur inside the logical schema of an IEDM, because it makes the model needlessly more complex. However, metadata which is relevant for interoperability should be standardised as well, either added to the physical schema of the IEDM or defined in a separate 'exchange metadata model'. The boundary between metadata and normal data is not always clear.
11. Versions of data, mostly time-related (historical), should only be used when really necessary and never as a form of metadata (e.g. keeping record of mutations on data).
12. An item must always refer to exactly one type, simply because that's the nature of items and types. The distinction between items and types should always be made clear. Type information is usually static, although dynamic examples do exist.
13. Key attributes should be meaningless numbers. A universal key management strategy has to be defined.
14. Only the most fundamental, recognisable and concrete (non-abstract) real-world objects should be represented by independent entities.
15. Relevant unstructured information should be integrated in an IEDM by linking it to the structured data (by using attributes with references to documents, pictures, etc.). The unstructured information itself is defined outside the models schema. One should standardise on the possible file types and formats.

Guidelines for multiple IEDMs

16. In a large environment a single all-enclosing IEDM is not feasible. Therefore, a subdivision into *multiple* exchange languages — each with a specific (limited) scope — will be necessary. These IEDMs have to form a hierarchy that expresses how they are related. This so-called "information interoperability domain structure" is a solution to manage enterprise-wide interoperability.
17. Besides not too large (see back), the scope of IEDMs should be not too small either, because that will result in a large number of information exchange standards. Another scoping guideline is to get the optimal *subdivision* of systems and information types into separate interoperability domains. IEDMs should be as autonomous as possible. They contain systems with much information in common, which is exchanged between them frequently.
18. The different IEDMs should be *compatible* to a certain degree, which makes translation of data between them easier. This is achieved by applying the same modelling guidelines to all IEDMs and by basing all IEDMs upon a "standardisation data model". This is a common glossary containing basic information types and structures.

6.3. Suggestions for JC3IEDM

The paper suggests several improvements for JC3IEDM. The major ones are repeated below.

1. Make clear whether the (future) JC3IEDM will only serve data exchange or will be used for wider standardisation purposes as well. In our view, JC3IEDM should support information exchange only. Besides that, an additional standardisation model is needed to assure compatibility with other IEDMs.

2. The intended scope for the future JC3IEDM is not (yet) completely clear and should be stated more explicitly.
3. The scope of JC3IEDM appears to be joint/combined NATO C3. This is too large. A better scoping of JC3IEDM, according to the interoperability domain approach, could entail the following actions: Split JC3IEDM into a Land C2 model and a generic standardisation model. Develop *separate* models for the functional areas of Air C2 and Maritime C2, both based upon that standardisation model. Synchronise with other domains (such as Intel and Logistics) to refine the scope. Set up a central board that co-ordinates (not: controls) all modelling work with respect to interoperability in NATO.
4. The logical and physical schemas of the JC3IEDM are almost equal now. The physical schema must be optimised for exchange, making it structurally different from the logical schema.
5. The volume of the JC3IEDM is too large (195 entities), making the model very complex and thus hard to comprehend and implement. Some of the other guidelines could help to simplify JC3IEDM.
6. JC3IEDM contains several structures which are quite static, in particular the OBJECT-TYPE tree. These should be removed or at least simplified.
7. The JC3IEDM is very generic, but also contains quite some details, such as regarding MEDICAL-FACILITY-STATUS and HARBOUR. Such detailed information types do not 'fit' inside this model, given its (wide) scope. Instead, some of the details should be modelled more generically, the other (more specialised) details should be put in another IEDM (e.g., in a Logistics or a Maritime C2 IEDM respectively).
8. JC3IEDM should also be made more 'balanced' in the sense of generic and explicit structures. Some parts are very generic (e.g. ACTION, MATERIEL), while other parts are rather explicit (e.g. MASS-GRAVE).
9. JC3IEDM contains some structures which might have been denormalised in the logical schema. In addition, the physical should have been denormalised even further.
10. The entity REPORTING-DATA and the 'loggable' entities linked to it, involve a kind of metadata. Apart from a few exceptions, the 'loggable' aspect should be removed from the model and solved at meta-level.
11. JC3IEDM contains some 'unreal' independent entities (e.g. AFFILIATION and LOCATION). It would be better to remodel them as (dependent) properties of other entities.
12. JC3IEDM data can only scarcely be related to unstructured information. More (optional) references should be included in the model.
13. Non-key attribute names should not include the entity name. This will increase the models readability.

References

[1] "Information Interoperability Domains", Eddie Lasschuyt, TNO-FEL (NL), 11 November 2003, paper presented at the NATO RTO SCI-137 Symposium on "Architectures for Network-Centric Operations", 20-22 October 2003, Athens (GR)

[2] "Memorandum of Agreement between the Multilateral Interoperability Programme (MIP) and the NATO Data Administration Group (NDAG) on the JC3IEDM", 4 February 2004, Mons (BE)

[3] "Overview of the C2 Information Exchange Data Model", MIP, 20 November 2003

[4] "The C2 Information Exchange Data Model (C2IEDM) — Main & Annexes", MIP, 20 November 2003

[5] "Overview of the Reference Model within the NATO Corporate Data Model", NDAG, 11 June 2004

[6] "A New Approach to NATO Data Management", Bert van Domselaar, ISSC, presentation at the Bi-SC AIS Conference, 10-12 March 2004, NC3A, The Hague (NL)

[7] "Designing Quality Databases with IDEF1X Information Models", Thomas A. Bruce, 1992

[8] "Integration Definition for Information Modelling (IDEF1X)", Federal Information Processing Standards Publication 184, National Institute of Standards and Technology (US DoD), 21 December 1992

Annex — JC3IEDM (Logical Schema)

See separate sheet (2 pages).

As explained in chaper 2, the current version of JC3IEDM is equal to C2IEDM Edition 6.1. The last one is actually printed on the sheet.