

A Collection of Papers Describing Concepts Developed Between 1994 and 2005 Under Contract for the British Army Toward their Goal of Fully Interoperable Application Systems

*Harry Ellis,
Little Twitchen, Burrington,
Umberleigh, Devon, EX379JU, UK.
02 February 2006
harryellis@acm.org*

Between 1994 and 2005 the prime author of these papers was working under contract to the British Army on an extensive search for means to make all their diverse application systems fully interoperable by 2010. A solution was found and adopted in principle. Implementation is in progress but It remains to be seen how closely this will follow the radical ideas involved.

This selection of four white papers traces the conceptual thinking behind this 'solution'. It begins with the most recent exposition at the first Semantic Technology Conference (2005) and then works back through three earlier papers that show how these ideas evolved.

“Removing Technical Bias from the Conceptual Modeling of Business Information” - Harry Ellis and Peter Nell - Published at The Semantic Technology Conference (San Francisco, March 2005).

This paper deals with the core principles that the British Army has evolved for information modelling. It was designed to bring these principles to the attention of the Semantic Web community in the belief that they might prove useful.

“Sharing Live Application Data in Real Time Across the Internet” – Harry Ellis – Published at The Enterprise Data Forum (Pittsburgh, November 2002).

This paper was the first published description of what is now perceived as the “global knowledge base”.

“Using the Quantum Data Model to Develop Shareable Definitions” – Harry Ellis – Published in “Conceptual Modeling for New Information Systems Technologies, ER2001 Workshops (Yokohama November 2001) - Springer LNCS 2465.

This paper was the first published description of the thinking behind the classifying scheme idea that is now perceived as the key enabling concept for a “global ontology”.

“A Solution to the Problems of Sharing Dynamic Information Between Systems that are Subject to Independent and Asynchronous Development” – Harry Ellis – Submitted to the BCS Journal (May 1999) but not accepted for publication.

This paper was the first exposition of the core principles behind the 'claimed and adopted solution' to the Army's problems. Sadly it failed to meet the exacting standards of the BCS Journal but is believed to have value as a record of significant ideas and useful work done.

Removing Technical Bias from Semantic Conceptual Modeling of Business Information

A New Language with Control over Subjective and Variable Classification

Harry Ellis and Col. Peter Nell

Directorate of Command and Battlespace Management - Capability Integration

The British Army

Shannon Building, Blandford Camp,
Blandford Forum, Dorset, DT11 8RD.

UK

harryellis@acm.org

ABSTRACT

This paper shows how the British Army is using semantic modeling to gain control over the information content of major systems. It concerns the sharing of information between allied military forces – information about things like terrain, equipment, capability, command structures, orders, targets, status and location.

It addresses one of the main causes of delay and overspend in the procurement of new information systems. The objective of this research was to find a way of modeling information that is entirely free of bias toward any software product or design. This freedom from technical bias is needed to enable the British Army to control the information content of systems where the design authority is vested in an external contractor. The paper shows that the key to such control lies in rules that govern the use of different types of information to describe and further categorize each type of thing.

Keywords

Information, Modeling, Semantic Web, classification.

INTRODUCTION

This paper describes key results from a ten year research endeavor driven by the need for information to be shared by a large number of systems procured at different times with different priorities each with their own different contracted design authority. A solution has been found through a new modeling language that is focused on units of information that are indivisible, self-contained and cannot be changed without creating a new version of the whole

*Copyright (200): Harry Ellis and Peter Nell,
Little Twitchen, Burrington, Umberleigh, Devon
EX379JU, UK.*

THE SITUATION FACED BY THE ARMY

In specifying and procuring battlefield information systems the British Army faces a unique challenge. Under British Defence procurement guidelines (Smart Procurement) the Army must specify its information needs without imposing any technical constraints on the contractor, such as data standards and database constraints. However, the Army must treat its information as a corporate resource because sharing of information is the key enabler of Network Enabled Warfare.

The challenge inherited by the Directorate of Command and Battlespace Management was to find a way to specify requirements which include information needs taken from a corporate perspective, but within the constraints imposed by Smart Procurement. This means that the specification of required information content must be:

- Analyzed and captured from a business/operational perspective rather than a technical/implementation perspective.
- Easy to understand by both developers and users, who must be able to validate the representation.
- As objective as possible, with all subjective interpretations and judgments removed.
- Represented in such a way that no technical implementation solution is presumed.
- Able to deal with part or all of the operational processes under consideration, independent of the wider context in which they occur.
- Able to focus on specific operational perspectives, without having to include wider issues which may cloud the analysis, unless these are germane to the perspective being analyzed.

Faced with this challenge we ran into serious problems with the widely promoted languages for modeling data and were forced to find some alternative.

THE NEED FOR A NEW MODELING LANGUAGE

It soon became clear that the difficulties we were experiencing lay deep in the original purpose for which the established languages had been created – ERWIN for database design and UML for software design.

We needed a new language designed expressly for defining information in a way that is equally meaningful to both the user community and the developers of information systems. We would have to start from scratch by understanding the way users think about information. Then we would ensure that the resulting concepts can be reconciled with established data theory and practice.

This paper is focused on just one of many reasons for this decision – that of “subjective and variable classification”.

Subjective and Variable Classification

From a business perspective it is obvious that the classification of things is not always fixed throughout their life but can change in response to events. There are two reasons for this:

- **Subjectivity.** This arises where the classification is a matter of opinion or observation. A business like the Army cannot just accept such classifications as fact. It needs to know “who said?” and must allow for competing versions from different sources.
- **Variability.** This arises where the facts on which the classification is based may change over time. Clearly the business may have a requirement for preceding versions to be retained until all significance is lost.

Awareness of each classification is shared through information containing reference to some extension of the parent class. Superficially these ‘categorizing references’ are hard to distinguish from other information that is purely descriptive. For example compare the phrase: “Fred is a soldier” with “Fred’s date of birth is 3rd March 1984”. Both of these express information that helps to describe the subject “Fred”.

The essential difference between these two pieces of information is that “Fred is a soldier” does more than describe. It also places “Fred” in the specific category known as “soldier”. The significance of this in an information model is that such categorization may carry specific implications of inheritance that apply only to the individual instance concerned. For example it may be that further information such as “service number” is needed to properly describe “Fred” because, while and only while we believe that he is a “soldier”.

To formalize this implication we need to define the two types of information differently. All we need for “date of birth” is to say that its value must be a date, that only one value for this date can be correct and that this ‘correct’ value can never change. Note that in an information model we are not concerned with provision for the correction of erroneous reporting as this is part of system design and lies on the developer’s side of the contractual divide.

For information like “Fred is a soldier” we need to say a lot more. First the model must say formally that this information expresses categorization. Then we need to distinguish this particular type of categorization from others such as “Fred is a male person”. Already we have two ways in which any Person may be categorized. These might be called “Person by military status” and “Person by gender” respectively. Clearly these are independent of each other. They are each represented by a distinct set of categories but both are constrained for use in the categorization of individual instances of Person.

Contingent Characteristics.

The insight that we wish to share in this paper is that information in the form of categorizing references has the profound implication that characteristics associated with the cited category become applicable to the subject while and only while the categorization is current and valid.

While this implication may seem obvious, it is not directly supported in the popular data modeling languages. It is of course true that languages like ERWIN and UML have great value in supporting the design of databases and all types of software. Our concern is that they do not allow business users to express requirements such as subjective and variable classification without specifying the mechanisms through which they are to be supported. Business users are not generally qualified to decide such detail.

It is also inappropriate for the users of a system to appear to tell the developers which mechanisms to use. For this reason it soon became clear that a new language was needed so that this type of classification with all its implications could be simply expressed as a statement of requirement that makes no assumptions about its implementation. The required new language has been in development for five years and is now widely used within the British Army. It is called (Corporate Business Modeling Language) or, more commonly, “CBML”. In the brief description of the language which follows certain terms are used with very precise meanings set out in a glossary that can be obtained on request from the authors. Such terms are highlighted by use of the italic form with bold type on first use.

CBML – an Outline

The primary function of a CBML model is to express rules that govern the types of information by which things are described. Where these things have individual identity they are known as *entities* and the types of information by which they are described are known as *characteristics*. Please note that in this paper the term “entity” is used with its true meaning as an individual thing and not as shorthand for “entity class”. Each of these characteristics defines a type of *information element* that is self-contained, indivisible and cannot be modified without creating a new version of the whole. This definition means that all information is broken down into elements that never change and can therefore be moved around at will.

The Developer’s Perspective

CBML is chiefly a tool for business users to help them specify the types of information (*characteristics*) by which things of various types are described. Army experience has shown that CBML enables this to be done effectively by people with no appreciation of the technology of information systems. However this result was not achieved without a deep understanding of what such definitions mean to the developers of systems.

There is a general assumption built into the language that each individual version of a *characteristic* must never be overwritten and may need to be retained for an unspecified period. For this reason each *information element* must be regarded as separate from the subject it describes.

Separating out the *information elements* in this way means that the value of each must be supported by references to:

- The subject entity it describes,
- The characteristic that the value represents,
- Its *provenance*. This *provenance* may include references to its immediate and/or ultimate source, its context of origin, confidence level, precision and the time window within which it is deemed to be valid.

The value of each *information element* comprises one or more *information fragments* each comprising either: simple descriptive data, categorization or a combination of such. The rules that govern this composition are an important part of the definition for each *characteristic*. The processes of creation, replacement and cancellation of such *information elements* are controlled by a set of usage rules that specify variability, multiplicity, optionality and the need, if any, to retain versions over time (time plurality) and/or from different sources (source plurality).

CBML places responsibility for identifying characteristics firmly on the user’s side of the contractual divide. With the information broken down in this way the developer does not have to worry about updating and its consequences. CBML tells the developer what characteristics are applicable to the instances of each entity class. It says nothing about the required data structure beyond the general assumptions about separation and immutability of information elements that are set out above. On this advice the developer is expected to provide:

- A persistent data store that can hold *information elements* with all types of composition that are allowed by CBML.
- An API and user facilities through which applications and end users directly can create versions of any *characteristic* provided only that the subject must be currently classified as an instance of the entity class or category for which the *characteristic* is defined.

The problems of synchronization that arise in a database purporting to represent a single variable state of a business are shifted from the point of input to that of retrieval. The designers of information systems need to appreciate that responsibility for judging the validity of ‘updates’ must lie within the system and not outside. The old adage of “garbage in – garbage out” must be replaced by an acceptance that the quality of input is a matter for decision based on the *provenance* of each piece of input. Each system must provide filters to select those *information elements* that are relevant to its purposes. In many cases it will be appropriate to provide further filters at the point of retrieval so that each user sees only the versions relevant to the task in hand.

Classification in CBML

The basic idea behind CBML is that the *characteristics* that may be used to describe each *entity* are determined by the way in which that individual *entity* is classified. It is also recognized that an *entity* may have many classifications and that these may vary over time. Combining these two ideas we find that the full range of *characteristics* needed to fully describe an *entity* is determined by the cumulative effect of all ways in which that *entity* is currently classified.

In ERWIN and UML it is assumed that the inheritance of an *entity* is determined on its creation as an instance of one chosen entity class. Each such class is defined with *intrinsic characteristics* (attributes) which define the types of information that are used to describe *entities* of that class. This is the conceptual foundation from which CBML has been developed. The chief distinctive merit of CBML is that it builds on this foundation to provide support for the additional concept of “classification that is subjective and/or variable”.

In the real world much classification of *entities* is subjective and/or variable as in the case of “Fred” discussed above. Each of these subjective and/or variable classifications is expressed by information citing some extension of the parent class. In CBML all such classifications are recognized as a very special type of information endowed with the important effect of extending the inheritance enjoyed by the individual *entity* concerned. To ensure that this implication is formally specified and readily visible on both sides of the contractual divide CBML provides the distinctive construct of *categorization*.

The CBML construct of *categorization* is designed for direct expression of a requirement for subjective and variable classification as explained above in the section of that name. Each type of *categorization* applicable to instances of an *entity class* is expressed through a *categorization rule* that specifies the set of categories that may be used in that case. Each *categorization rule* specifies a type of information. As such it has much in common with a *characteristic*.

The similarity of *categorization rules* and the rules that govern purely descriptive information has been the cause of much debate within the British Army. The issue of contention is whether it should be possible for fragments of information that express *categorization* to be combined with those that are purely descriptive to form a single *information element*.

The view taken here is that it would be unusual for a combination of this type to form a unit of information that is “self-contained, indivisible and cannot be modified without creating a new version of the whole”. However such “complex *information elements*” could occur in cases where the business requires each *categorizing reference* to be supported by other references and/or textual notes that provide some sort of justification. It is for this reason that we define each *categorization rule* as specifying one of possibly several types of *information fragment* within the definition of a single *characteristic*. This structure includes the limiting but more common case of *categorizations* that require no support beyond the standard *provenance* and thus form a complete *information element* on their own.

In database designs, support for subjective and/or variable classifications is usually provided in one of the following forms:

- An attribute constrained to an enumerated domain,
- An ‘is a’ relationship,
- A sub-type discriminator.

While such constructs can be designed into a data model, they do not exist as regular concepts with a formal definition and distinctive notation. This means that the implied extension of classification and its consequences is not readily visible in the model. A common way round this is to express such requirements in the form of ‘business rules’ or function specifications written in natural English. We reject this approach due to the persisting lack of any suitably rigorous standard and consequent dependence on the linguistic skill of the modelers.

The concept of *categorization* described above deals effectively with all classifications that are subjective and/or variable. However it does not remove the need for *entity class specialization*. This more basic form of classification has a different purpose and different implications that are also directly supported by CBML as explained below.

Although CBML was created primarily for users to express their information needs we have also ensured that it is equally effective in showing what is proposed for a new system or already available in extant data. This is necessary so that CBML can support a two way dialogue between the stakeholders in a system and its suppliers. There is little point in articulating requirements for information if these cannot be compared, in user meaningful terms, with what exists or is on offer. This means that CBML must also have the capability of expressing the more restrictive forms of classification that are used in system design.

This capability is provided through the concept of *entity class specialization* which is supported in CBML for expression of hierarchic taxonomies in which sub-classes inherit the definitions of all super-ordinate classes. Under such *specialization* the inheritance applies to all instances of a *sub-class*. This familiar and well understood concept is quite different from the distinctive CBML concept of *categorization* where the inheritance applies only to individual instances which have a current and valid *categorizing information fragment* citing a particular *category* as described earlier in this section.

CBML allows each class and sub-class to be specialized in several different ways to meet the needs of different business perspectives. Each different basis for *specialization* of an *entity (sub-)class* is represented by a distinct *entity class specializing scheme* with rules that govern the specification of further member *sub-classes* by the business as and when required.

Semantic Library

In CBML the sets of *categories* invoked by *categorization rules* form what we call a *semantic library*. This is developed by the business community to reflect changing needs for the *categorization* of individual *entities* in the normal run of business. This means that it is growing all the time and therefore cannot be frozen into a model under any sort of configuration management.

The *semantic library* does not of itself “do” any *categorization*. It just shows the *categories* that the business currently requires to be held available for use when its users create *information elements* in accordance with any of the associated *categorization rules*.

The *semantic library* is actually a sort of reference database with facilities through which suitably responsible users can extend the range of *categories* in a set. To ensure that this is done in a disciplined way each *category set* is defined with:

- A *scoping reference* to an *entity class*. This determines the type of subject to which its member *categories* can be applied.
- Rules called *category descriptors* that specify how the business will describe new *categories* as and when they arise. Working within these rules the business can extend the taxonomy in a disciplined way without any further conscious modeling.

The main role of the *semantic library* is to hold the specification of *contingent characteristics* defined for individual *categories* as already explained in the subsection of that name. Such *characteristics* are called ‘contingent’ because they become available for use in the description of an *entity* only if and while it has a current and valid *categorization* citing the *category* for which they are defined. In this respect they differ from the *intrinsic characteristics* that are defined for an *entity class* and apply to all its instances.

In the earlier section on “Subjective and Variable Classification” we introduced the simple case of “Fred” as an individual instance of (the *entity class*) Person with current *categorization* as a “soldier”. The “further information such as “service number” mentioned there is an example of a *contingent characteristic*. In CBML it would be specified in the semantic library as part of the definition of the category “soldier”. In CBML any type of characteristic, including those that incorporate a categorization rule, can be specified for any category. This means that it is easy to express rules for categorization that become meaningful only in respect of entities that have some other categorization already in place. For example it may be that, while and only while “Fred” is a “soldier”, he must be further categorized by “rank”, “trade” etc.

This concept of *contingent categorization* is a major contributor to the effectiveness of CBML because it relieves the modeler of any need to impose the straight-jacket of *entity class specialization*. This more traditional concept can thus be reserved for those occasions where CBML is used to articulate the information content of extant data structures in which the concept of *categorization* is not present.

Finally we need to consider what *contingent characteristics* mean for the developer of systems. It was said earlier in this paper that the developer is expected to provide both a persistent data store and an application/user interface. All *characteristics*, whether intrinsic or contingent, need not have any impact on the design of the persistent data store. Because the *information elements* are separate from the subject it is perfectly feasible to design a database in which the semantics of each is expressed through reference to the relevant *characteristic* rather than by the table in which it is held.

With such a database the only effect that adding a new *characteristic* has on system development is at the application/user interface. For those *characteristics* that are purely descriptive even this effect is minimal. The effect is greater for those that include *categorization* because there is need to ‘switch on’ the associated *contingent characteristics*. Even here there is much scope for generic design that can respond to development of the *semantic library* as it happens.

EXAMPLE

Consider the situation where a vehicle is adapted to support ambulance services. Details of the vehicle have already been recorded including its registration number, model, engine size, fuel etc. as required to ensure the necessary logistic and maintenance support. While the vehicle is adapted for use as a field ambulance there is a further need to maintain information on the type of casualty it can handle and the state of medical supplies on board etc.

In CBML this requirement is easily expressed. Types of information relevant only to the role of “field ambulance” are defined as *contingent characteristics* for the *category* “field ambulance” within a *category set* called “Vehicle by operational role”.

The terms under which the business wants to regard individual vehicles in any such role are expressed as a *categorization rule* within the definition of a *characteristic* for the *entity class* “Vehicle” or one of its specializations.

The expression of this requirement in CBML can be presented in either graphic or verbalized forms which are fully equivalent. The verbalized form can be generated automatically from a diagram drawn in Microsoft VISIO using a template designed for this purpose. The software used for this is known as “CBML Lite” and can be made available on an “as seen” basis.

Requirement Specification in ERWIN

In ERWIN the modeler is faced with a choice. Should the “field ambulance” role be represented by attributes that are conditional on the value of a “role” discriminator column or should there be just a foreign key pointing to a row in a separate “Field Ambulance” table?

A design choice like this is not difficult for an experienced database designer provided there is a clear statement from the ‘client’ as to the number of different roles and the attributes involved for each.

Now consider the problem faced by a business user with no experience of database design trying to define such information as an expression of requirement. How can such a person make decisions of that sort? It may not be known how many roles there are going to be. No one outside the contractor's design team can understand the full implications of either approach. Whatever choice is made the developer may disagree and proceed to do something different.

We have no quarrel with ERWIN or with the "relational model" in general when used in the role for which it was designed – i.e. for the design of databases. Our concern is only that it cannot be used for direct expression of requirements which include classification that is subjective and variable without making decisions that rightly belong on the developer's side of the contractual divide.

It is however possible to generate a reasonably efficient relational database design from CBML automatically. This process relies on a set of generic design decisions on the best way in which to implement any instance of each CBML construct. Any such process is a design activity that also lies on the system developer's side of the contractual divide.

Requirement Specification in UML

In UML the problem is different but no easier to resolve. A naïve appreciation of UML would suggest that the "field ambulance" role should be represented by an object class. This would mean that the object representing a vehicle adapted to this role would need to be linked to another object containing information about that vehicle in its role as an ambulance. This link, i.e. the categorization, would become a data element within the "Vehicle" class with no assurance that its categorizing role and implications will be understood and supported in the resulting system.

Another problem is how to express the need for an extensible range of categories. The category set called "Vehicle by operational role" would have to be represented by an object class. Each category in this set would then be represented by an object of this class. So the simple concept of "field ambulance" role would seem to be both an object of one class and a class in itself.

We know of one way in which this can be expressed in UML but this depends on a UML construct called "powertype" which is rarely used and unfamiliar to most UML practitioners. It is also very confusing for anyone other than a trained system designer.

We have no quarrel with UML in the role for which it was designed – i.e. for the definition of system artifacts. Our main concern is that it is unsuitable for direct expression of requirements that include subjective and/or variable classification. Any attempt to use it in this role demands a level of design awareness that cannot be expected of business users in general.

It is however possible to generate UML from CBML automatically. This process relies on a set of generic design decisions built into a small number of "foundation classes" that are specified with functionality appropriate to each CBML construct. The resulting interface classes would provide an API that is guaranteed to support the required information content and to be compatible with a relational database generated from the same CBML. Any such process is a design activity that lies on the system developer's side of the contractual divide.

CBML AND THE SEMANTIC WEB

Although peripheral to the main driver behind CBML we find that the resulting understanding of *information elements* and *categorization* provides insights that are highly relevant to the development of the Semantic Web.

This is not surprising since they both entail the sharing of information with meaning that is formally defined. For the British Army this sharing is between heterogeneous application systems across a flexible network. For the Semantic Web the sharing is between any system anywhere.

The key breakthrough for the Army was the realization that the information must be broken down into elements that are self-contained, indivisible and cannot be changed. The practical considerations that drove this breakthrough apply with even greater force within the Semantic Web.

Flexible Taxonomy

The first of these practical considerations was a realization that no simple hierarchic taxonomy could be imposed on the participating application systems. We found this within the British Army and it is even more true across the world at large. This problem is overcome in CBML by the way the *semantic library* is designed as a disciplined but open-ended source of class extensions as explained above in the sub-section of that name.

This design allows any user to augment an existing class or class extension (*category*) with a new class extension at will without having to negotiate with any other user. Information based on such class extensions is integrated with all other information in a standard logical data format. As new class extensions are introduced, any user can either access or ignore the new information at will.

If CBML, or at least some of the principles expressed in this paper, were widely used within the Semantic Web its use would be eased because:

- Any need for pre-negotiated agreement would be limited to a small number of highly generic *entity classes* such as Person, Location, Action, Materiel, Administrative Instrument etc. All more specialized concepts would be covered by class extensions (*categories*) within a global *semantic library*.
- The administration of this library would be trivial because of the small number of *entity classes* to be

agreed and the freedom granted to individual users to extend these classes at will by creating their own *category sets*.

- While a measure of local agreement on such *category sets* would be beneficial to those participating in such agreement, all users would be able to access and create such information. There would be no adverse impact on any user because they would each see the web through the provenance filters they choose to specify.
- The semantics of such extensions including *contingent characteristics* could be defined and understood by actual users without design skills.
- The information could be processed without any need for harmony of design across participating systems.

Provenance

A second practical consideration that drove the British Army to develop and then adopt CBML was the fact that much of the information used by commanders in the battlespace, is subject to conflicting evidence - a phenomenon known as “the fog of war”. This led to the need for all information to be linked in some way to its *provenance* so that judgment of its relevance can be made independently by or on behalf of each user at the point of retrieval.

Clearly this is a vital component in the Semantic Web with its open nature and absence of control over participants. Any single *provenance* may be shared by more than one *information element* and it does not matter how the link is established. What matters is to establish a standard for the *provenance* to which all participants would conform. We see it as significant that leading relational modelers have been moving in the same direction with what Date and Darwen call “sixth normal form” [1] There is also the NATO Joint Consultation Command & Control Information Exchange Data Model (JC3IEDM) which is an information exchange data model developed for the Multilateral Interoperability Program. [2] This model includes its own *provenance* standard under the name of “Reporting Data”.

Distribution without Synchronization

A third consideration for the British Army was the practical impossibility of synchronizing data across its network. This impossibility is chiefly because the sharing must continue to function effectively while parts of the network are under attack and suffering damage.

For the Semantic Web we find the same impossibility of synchronization for other reasons including its openness and absence of central control.

The only way to achieve such sharing without synchronization is for the information to be broken into elements that cannot be changed other than by replacing them with a new version of the whole. This whole issue is explored in a paper published by Wilshire Conferences in connection with the Enterprise Data Forum November 2002 [3]

The Potential Role of CBML within the Semantic Web

The three issues discussed above all demand some way of getting the business users of information to identify the basic units of information that are self-contained, indivisible and cannot be changed without creating a new version of the whole. The British Army has spent five years developing CBML to do just that.

We see the vision of the Semantic Web as a global version of what we have been trying to do for the last ten years. We now feel ready to share our results in this wider arena. The placing of such information on the web would be controlled by each contributing user through rules that reflect their own intent. Each contribution could take any of a number of forms.

For example:

- Publishing would make the information available for anyone to access through some search process.
- Contract distribution where information of specified types is ‘sent’ over the internet to a consortium of users in accordance with their particular needs.
- Subscriber mode where potential users sign-up, with or without payment, to receive specified types of information.

To appreciate the full benefit of the discipline imposed by CBML it is necessary to visualize the actual data that would be shared. In essence all shared information would comprise self-contained *information elements*. Each of these would have its own:

- Subject reference,
- Meaning reference (*characteristic*),
- Value,
- *Provenance* detailed with references for “source” and “context of origin” which can be used to obtain further detail.

The value of this concept is greatly enhanced with the concept of Universal Resource Identifiers (URIs) promoted by W3C [4]. All of the above references could be expressed as URIs with the benefits of global uniqueness and accessibility.

Information in this form would be fully meaningful to any recipient because:

- The subject is positively identified by its URI.

- The “meaning URI” provides access to a full semantic definition including relevant links within the relevant *semantic library* which could identify the whole-life classification of the subject should this not be known by the recipient.
- The value, in standard string format, would have logical composition obtainable from the “meaning URI”. Some of these global ‘pointers’ could refer to widely accepted standard data types or to specializations of these that are specified by individual users. Others would cite the CBML concept of *categorization* and thus trigger inheritance implications associated with the cited *category*.

The *provenance* would enable each participating system to decide which *information elements* are relevant to its own purposes. Each system would be able to store and manage its own copies of such information with complete integrity while retaining this *provenance* detail to drive its own archiving processes. Of course the *information elements* that any such system chooses to store may be superseded by later versions and each system remains free to make best use of such ‘updates’ as and when they become available.

CBML and RDF

We in the British Army welcome RDF, W3C’s Resource Description Format [5], as a sound foundation for the Semantic Web. We find that this standard is easily used for full and modular expression of all information defined in CBML including:

- The definition of CBML itself.
- Any part of any *semantic library* including *category sets*, their member *categories* and *contingent characteristics*.
- Any collection of subjects with their initial whole life classification including any *specialization*.
- Any collection of *information elements* about any subject including those that express subjective and/or variable classification.

CBML IN USE

CBML is now used within the British Army to underpin all information requirement specifications. It is being used to specify:

- The information content of the BOWMAN Common Persistent Data Store. “BOWMAN” is the British Army’s high capacity data radio system, currently

being rolled out into service. The associated Common Persistent Data Store is the repository for all shared battlespace information.

- The information interface requirement for all battlefield information systems. This covers requirements for the exchange of information between the British Army’s own information systems and those of allied forces.
- The information content of all British Army formatted text messages – predefined, automated messages used for reporting purposes.
- All information content within the British Army’s enterprise architecture – “The Command and Battlespace Management Architecture”.
- All information content within British Army’s battlespace information system application user requirements.

ACKNOWLEDGMENTS

The British Army: Lt Col Chris Bailey, Lt Col Mark Thurlow, Maj Richard Garbutt, Martin Richley, Peter Lawson, all of whom have played a major role in the development of CBML and its exploitation.

Skellbase: Dr Ken Allen, who has been a major force in its development from the start.

Salamander Organisation: Toby Sumpter, who is leading the development of architectural tool support for CBML.

REFERENCES

1. “Temporal Data and the Relational Model”, C.J.Date, Hugh Darwen & Nikos A. Lorentos, Morgan Kaufmann, ISBN: 1-55860-855-9.
2. Multilateral Interoperability Programme, www.mip-site.org.
3. “Sharing Live Application Data Across the Internet – A new Concept in Data Storage”, Harry Ellis – British Army, Enterprise Data Forum (EDF2002) download from www.wilshireconferences.com/EDF2002/Ellis_Harry.pdf.
4. World Wide Web Consortium, www.w3c.org
5. . Resource Description Framework (RDF) Concepts and Abstract Syntax, W3C Recommendation 10 February 2004 obtainable from www.w3c.

Sharing Live Application Data in Real Time Across the Internet

Harry Ellis

British Army

Army Directorate of Information, Faraday Building, Blandford Camp
BLANDFORD FORUM, Dorset, DT11 8RE,
UK
harryellis@compuserve.com

Abstract

This white paper describes how live data produced by an application could be made accessible at will to any user of any application system anywhere in the world. It describes a new form of data server that would provide a service of secure persistent storage with automatic rule-based distribution to similar servers across any wide, varied and dynamic network including the internet. The paper introduces a concept known as “quantization” that allows asynchronous transfer between servers to ensure high overall resiliency. Key features of an enabling technology are explained. These include a new language called CBML to show the use of common data within different user perceptions, a matching XML standard and rule-based generation of bespoke APIs. There is also a summary of the impact on various aspects of information system management. Constructive comments are sought with a view to serious exploration of the potential market for the envisioned product.

1. Introduction

The vision described in this paper is a practical one based on eight years research and development undertaken by the British Army. The key driver for this work was a directive issued by the Executive Committee of the Army Board in 1987. This directive required that all information systems be fully interoperable by the year 2010. It soon became apparent that this goal was unattainable through the prevailing technology for the design of on-line databases. For this

reason it was decided to take a fresh look at the technology with data sharing as the overriding consideration.

This work has shown that the required interoperability could be achieved but only through a new kind of data server. This would be designed to provide a service of secure persistent storage with automatic rule-based distribution to similar servers across any wide, varied and dynamic network including the internet.

The main reason for this conclusion was the degree of cultural and technical change required to fulfill the goal of full interoperability. Initial reactions from application developers have made it clear that the novel ideas required to meet this goal would have to be encapsulated within a ‘black box’ behind familiar interface technology. This product would have to be designed for introduction on a modular basis to give unlimited scalability with proportionate return for investment of cash and other effort. It would also be necessary to provide high quality tools for the specification and flexing of services so as to reduce the need for training to an absolute minimum.

The initial challenge was to achieve these goals across the sort of flexible WAN with mobile nodes, high levels of security and high resilience that is required by military forces on deployment. Having solved this challenge at a conceptual level it soon became apparent that the same technology would be equally effective for commercial and other purposes with global access through the internet.

2. The vision

The vision outlined in this paper foresees a world in which the designers of any application system would have the option of a new and more flexible form of persistent storage for information. Selection of this option would ensure that live data produced by the application could be made available at will and on any basis to any user of any application system anywhere in

the world by the simple process of writing an appropriate contract.

2.1 The product

The key to fulfilling this vision is a new form of data server that we shall call a Data Service Point (DSP). This new product would combine the best of reliable persistent data storage with software to provide contracted services with all necessary replication, distribution, access control and back-up performed automatically in accordance with rules. The new concepts and standards required to achieve this would all be encapsulated within the product behind familiar interface technology. Each DSP would be installed as a server on a LAN. It may also form part of a wide area network and/or be accessible through the internet.

Practical considerations demand that such DSPs be designed as modules that collaborate with each other on a peer-to-peer basis to form a data service network of any shape or size. This would ensure that they could be introduced into any enterprise on a piecemeal basis alongside current storage technology. The service provided by these DSPs would have substantial added value as compared to a conventional database server. It would relieve the developers of client application systems of all need to consider requirements for security, recovery, distribution and/or any form of interface with other application systems.

2.2 Automatic distribution of data

The most distinctive added value offered to the client application systems is that the service would include automatic distribution of the most up to date versions of selected data to all DSPs at which prospective users of that data are registered. This distribution would be transparent to originators and beneficiaries alike. It would be achieved in a way that would ensure high security and greatly simplify the management of dynamic data.

The developers of an application system would then have the option of using a DSP as an alternative to conventional database for the persistence of some or all of the information it creates. Information placed in the care of a DSP would form part of a pool of information that forms an enterprise resource controlled by the originators of information rather than by any one application system.

Each contributing application system would provide its own distinct information service by providing means by which its users may draw on and contribute to this information pool in a particular way. Thus it would provide one of many intelligent paths between the user and the overall body of relevant information arising from many sources through many application systems. The potential for access to such information would extend to any application installed on any client

processor with local connection to the same DSP. Actual access to specific instances would depend on the real-world context of origin and be subject to permissions granted by an appropriate authority.

Where many DSPs are installed as peers across a WAN they would collaborate by making best use of available communications bandwidth to pass copies of new information to each other in asynchronous fashion so as to optimize the distribution of information. This would be the prime means by which information would be made available to applications running at physically remote locations.

Because the distribution is asynchronous this sharing is readily extended across the internet to DSPs and their clients on other WANs and to any free-standing client with an internet connection.

2.3 A data infrastructure

All services provided by a DSP would be rule based with the rules expressed in contracts that can be created, modified and switched on and off at will without specialist IT skills and without change to any installed software. All installation and management functions including client registration, the specification of contracts as well as direct user query and data entry would be executed through a single software package that could be installed on any recent operating system and be connected to any DSP. We will call this software package "DSP Manager".

Natural growth in the numbers of DSPs and their use by application systems within an enterprise or industry grouping may reach a critical mass at which it will be perceived as a *data infrastructure*. This situation would be reached when a useful proportion of the data needed to satisfy a new information need is readily available as a by-product of existing application system clients already using the service. At this point the *data infrastructure* with its constituent DSPs will become the automatic choice for all but the most specialized requirements for data persistence.

A *data infrastructure* like this would serve as custodian for live application data on behalf of the enterprise as a whole. As a natural consequence of events it will become aware of the information needs of every user role and the locations at which these needs are to be met. It will use this knowledge to direct automatic distribution of data across a network of DSPs so that the information needs of each user role can be met locally at any chosen location.

Security of all data placed in the care of any DSP will be assured in accordance with wishes declared at the point of entry for each piece of actual information. The way this responsibility is split between an application system and its users is a matter of application system design that is independent of the way in which the *data infrastructure* is configured.

Each element of information will be tied to a source and a context of origin. Other users will be allowed to see and/or add information to a context of origin only in accordance with permission granted by its owner.

The owner of a context of origin may also delegate its rights of ownership to other users so that the security framework can be adapted in response to local knowledge and circumstances. These delegated rights may allow such users to create their own new sub-contexts. This would give them free choice between keeping tight control over their information by placing it within one of their own sub-contexts or making it more widely available by placing it within any broader context to which they have been given write access.

3. The core idea - quantization

The vision outlined above can not be realized without a fundamental shift in the way in which data is perceived and stored.

The core idea that makes the vision possible is that information about each real-world object should be broken into self-contained information elements that can never change, are free-standing and completely independent of each other. The ‘no overwrite’ rule implied in this core idea ensures that any *information element* can be replicated and distributed at will without any risk to integrity.

The prevailing notion differs in that data is arranged into records such that each record contains a set of variable attributes of some real-world object or part thereof. This notion is rejected as being both simplistic and the underlying cause of the many problems of application integration, data management and security that are faced today.

The traditional concept of a database comprised of cross-related tables is regarded as applicable only to the physical storage of individual *information elements*. For this reason both the concept and the associated technology of relational databases should be hidden from both the users and designers of application systems. People in these roles should be concerned instead with an *information base*. This would be populated with freestanding *information elements* each containing a piece of self-contained information about a single real-world object.

This general approach to the structuring of stored information is called “*quantization*”. This name was chosen because these independent *information elements* each represent one of many possible values for one of many *characteristics* of one real-world object. The set of such *information elements* about one and the same real-world object provides a quantum description of that object because it shows a superposition of all known states of all applicable *characteristics*.

The collection of such *information elements* for all *characteristics* of all real-world objects within a given

scope forms an *information base*. An *information base* together with its supporting *master index* and *semantic library* we shall call a *quantum data store*.

3.1 A quantum data store

Each of these *information elements* represents one version of the value of one *characteristic* of one real-world object. For this reason the *information base* is supported by a *master index of real-world objects* and a *semantic library*. Every *information element* has both a *subject pointer* that identifies the real-world object it describes and a *meaning pointer* that identifies the applicable definition of meaning within a *semantic library* as shown in Fig 1. below

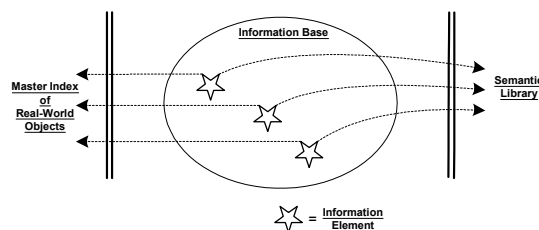


Fig 1. Information Elements

Master Index: This is a simple list of real-world objects each represented by a globally unique identifier (*GUID*) and nothing else.

Semantic Library: This contains definitions of meaning for all *characteristics* each distinguished by a *GUID* and supported by appropriate descriptors.

Information Base: This contains physical images of a non-exclusive subset of *information elements* from within its parent *quantum data store*.

3.2 The structure of information elements.

To ensure the independence of these *information elements* it is necessary to ensure that the *useful content* of each is associated with a standard *control wrapper* comprising five control elements as shown in Fig 2.

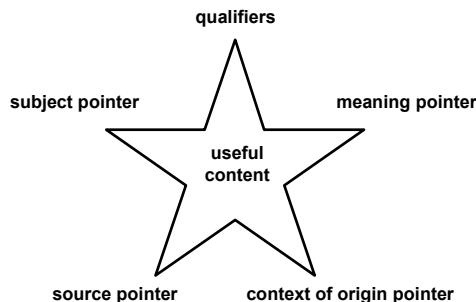


Fig 2 Control Wrapper for Information Elements

These five control elements combine to provide meaningful distinction between the vast number of

information elements and provide the basis on which selection is made.

Subject Pointer: This is comprised solely of the *GUID* that identifies the real-world object described. This pointer is necessary because the *useful content* is not physically tied to any representation of the real-world object concerned as it would be in a relational database.

Meaning Pointer: This comprises the *GUID* that identifies the type of information (*characteristic*) expressed by the *useful content*. By way of example, in the case of a person this would serve to distinguish the *information element* for ‘date of birth’ from that expressing ‘surname’.

Source Pointer: This comprises the *GUID* of a real-world object that is deemed to be responsible for the accuracy of the *useful content*. This is needed so that retrieval by an application system and/or user can be restricted to those *information elements* that were created by sources selected as appropriate for the purpose of that retrieval.

Context of Origin Pointer: This comprises the *GUID* of a conceptual object chosen by the creator of the *information element* to represent the operational and other circumstances under which the *useful content* was created. This serves to place each *information element* permanently within a partition of the *quantum data store* that can be accessed only by those users for which access permission has been granted.

Qualifiers: These comprise a group of optional descriptors that serve to further differentiate between different versions of the same *characteristic* for the same real-world object. This group includes descriptors like “as at time”, “reporting time” and “confidence level”. These are provided so that individual application systems and/or users can further refine the selection of versions relevant to each different purpose.

3.3 Create, retrieve, update and delete rules

The most distinctive feature of *information elements* is that they never change and can never be logically deleted. In spite of this, a data store made up entirely of such *information elements* is able to reflect faithfully all observed changes in the real world including those derived through some data process. This is because the description of any given real world object is represented by an open set of such fixed *information elements* that is extended whenever new observations are received.

Rules for create, retrieve, update and delete operations are normally thought of as applying to system objects. In this case it is more appropriate to consider them as applying to real-world objects as follows:

Create: When a new real-world object is first recognized within the *quantum data store* it is necessary to allocate a *GUID* and thus notionally place

it within the *master index* of all real-world objects. This process will normally be coupled with the creation of at least one *information element* to provide a real-world key for use in subsequent direct access.

Retrieve: Retrieval of descriptive information about a real-world object requires knowledge of its *GUID*. This may be obtained either from the *subject pointer* of an *information element* used as a real-world key or from a reference within the *useful content* of an *information element* that has already been retrieved. The normal pattern of retrieval is a two stage process. The first stage is to nominate selection criteria, obtain a hit list of matching candidates with real-world keys and/or other *information elements* and then use this to select one *master index GUID*. In the second stage this *master index GUID* is used as a handle to retrieve associated *characteristics*. Navigation to related real-world objects is achieved by taking-up their *master index GUIDs* as quoted within *referential characteristics*.

Update: Update of the description of a real-world object is performed by creating new *information elements*. Qualifiers such as ‘as at time’ are supplied to distinguish the new version from previous versions of the same *characteristic* for the same real-world object, from the same source and with the same context of origin. These qualifiers are needed for use in the filtering applied in subsequent retrieval operations to reflect the needs and preferences of each user purpose.

Delete: Nothing can be logically deleted from a *quantum data store*. However the originator of any *information element* may set an advisory obsolescence flag to indicate that, in the opinion of its creator the associated *useful content* is to be disregarded. This ‘no delete’ rule applies to the *quantum data store* which is a logical collection of *information elements*. The persistence of any physical instantiation of an *information element* in any *locality database* is a different matter controlled by a rule-based archiving process. In appropriate cases where it is judged that the out-dated values of a *characteristic* have no value or where the archive facilities are unavailable these contract rules may specify ‘archive to air’ and the information would be lost.

For each *characteristic* of a real-world object there is a distinct life history comprising an unlimited number of such versions each represented by a distinct *information element*. The life history of each *characteristic* is independent from that of all other *characteristics* and from that of the real-world object it describes.

3.4 Data ownership

Data structures within the envisioned *quantum data store* would not reflect any presumption about the purposes for which the information may be used. This

has a radical effect on the concept of data ownership as commonly understood..

Information about a real-world object may come from many sources beyond the control of any single owner. For this reason the state of any real-world object as a whole is not stored but is compiled on demand according to need so that the concept of a single owner becomes meaningless.

This compilation is undertaken by the infrastructure as part of the retrieval process. This is done on behalf of the user or application system responsible for initiating the retrieval and is based on rules invoked as part of the retrieval call. Neither the rules nor the compiled state of any real-world object are of any concern to the various creators of the information thus selected.

3.5 Information contained at data service points

Each DSP contains a physical instantiation of a non-exclusive subset of the *information elements* that comprise the logical *information base* of a single *quantum data store* as illustrated in Fig 3. below.

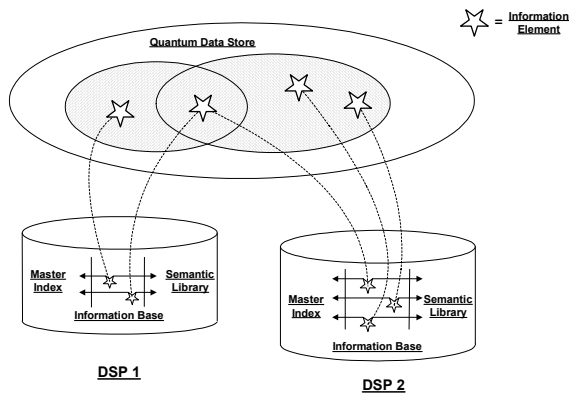


Fig.3 Data content of Data Service Points

The significant point here is that each DSP contains one of many possible instantiations of the logical quantum data store of which it is the local instantiation. What distinguishes these physically separate instantiations is that they each contain a distinct selection from the complete logical set of *information elements* that make up the *quantum data store*.

This selection is tailored to meet the anticipated needs of a set of registered users. It is also subject to continual change with the addition of new *information elements* that the rules deem to be relevant and the removal of old ones that the rules deem to be obsolete. It is thus a dynamic rule-based subset of the parent *quantum data store*. The completeness of this subset is of course subject to the practical limitations of prioritized use of the available communication links.

4. The Arguments

The purpose of this section is to provide an outline of the intellectual justification for quantization and the related ideas.

4.1 Update without overwrite

The core idea of *information element* as set out in section 3. has roots in the concept of *elementary message* advocated by Langefors in 1973 and earlier. This concept, also known as *e-message* is defined in [1] as being comprised of:

- Identity of object (subject pointer)*
- Kind of property (meaning pointer)*
- Specification (useful content)*
- The point in time (qualifiers).*

The labels in brackets refer to the equivalent term in the *quantum data store*. It can be shown that the *information element* concept of the *quantum data store* can be seen as an *e-message* extended significantly by further control elements that are needed to deal with multiple sourcing (*source pointer*), partial distribution (*context of origin*) and competing versions (*further qualifiers*).

The overall philosophy of the *quantum data store* is a bit like taking J. Bubenko's vision of a knowledge base (KB) and updating it to exploit the massive power, storage and global communications that have evolved over the past twenty years. Bubenko said in an invited paper presented in 1983. [2]:

"The KB should also view its domain in a time perspective and not restrict its knowledge only to the 'current state' of the UoD. ... The KB should never 'forget' anything." He also said:

"There is no concept of 'modifiable store' in the KB and the concepts of updating and deletion, consequently, do not apply".

In a quantum data store the concept of update is not just excluded like this. Instead, update is supported at the application and user interfaces while nothing is ever overwritten in the data store. This 'trick' is achieved through the API which presents a 'latest' or otherwise appropriate description of a real-world object compiled from an appropriate selection from the locally available range of stored versions of individual *characteristics*.

4.2 Distribution without synchronization

Knowledge about the state of each *characteristic* of any real-world object may comprise a number of independent and possibly conflicting observed or derived values. Observed values represent subjective real-world experiences of an observer, sensor or measuring device that occur at various points within the space-time continuum and are logically independent of each other. Derived values are the result of a particular instance of a particular information process which also

occurs at a specific point within the same space-time continuum.

The criteria for selection of any particular version as the current state of a particular *characteristic* depend on the purpose for which this current state is required. For this reason it is necessary to defer such selection until the point of retrieval where it will not interfere with the process of selection for any other purpose. This is made possible in a *quantum data store* by preserving all observations together with sufficient detail of the associated circumstances to enable selection of those relevant to any particular purpose.

In the envisioned *quantum data store* all need for synchronization of updates across the many DSPs is avoided by replicating only those elementary units of information that represent one state of one independent *characteristic* of one real-world object.

4.3 Comparison with the relational model

The state of the art for logical data design has long been dominated by the relational model. For this reason comparison with the relational model is one important way in which the principle of *quantization* must be judged.

In spite of popular teaching that each table should represent a class of real-world object, the idea of using a table to represent a single *characteristic* is found wherever a system has a known interest in a set of values over time. Where *quantized data* differs is that this approach is used for all *characteristics* as a general principle to include a range of reasons for storing more than one value of a single *characteristic* at the same time.

Taking the possibility of multiple values as a general principle affects the behavior of the data store as a whole but does not challenge the validity of relational principles in respect of physical data design.

A relational database is generally perceived as a body of data that is organized in accordance with relational principles and is controlled by a single design authority. Any correspondence between such system objects and any objects in the real world occurs only in the mind of the designer of 'the system'.

When data is *quantized* this 'system centric view' is replaced by a 'real world view' populated by multi-faceted images of real-world objects with *characteristics* that depend on classifications that may change over time.

It has long been recognized that there should be a single conceptual schema to which all application systems subscribe. This cannot be achieved in large or complex enterprises unless information users are prepared to compromise on how the data is defined. *Information elements* within a *quantum data store* permit different users to hold their own perceptions of information while sharing common data. This gives

quantized data a strong practical advantage over data based solely on the relational model.

The methods and tools that dominate the market at present have grown to meet the needs of highly skilled system specialists working in a close-knit project community of people with the same mindset. For this reason low priority has been given to understanding the meaning of information at the business level. All the emphasis is on structure to suit the declared aims of a single information system. This is for the very good reason that the structure of data has to be formally defined for any software to work. The resulting data structures tend to be a mix of the simplistic and the obscurely clever with rather vague indications of the business meaning of the data they contain. It is this that obstructs integration more than technical differences that can be resolved by formal rules.

4.4 Resiliency through localization of risk

The growing demand for resiliency requires that change, loss or damage to any part of the network must not affect the integrity of the data that remains. It also requires that this remaining data shall be sufficient to sustain core business functions and enable full recovery.

This is achieved in a *quantum data store* by writing the rules to exploit the scope for planned redundancy that is inherent in the core idea of *quantization*. All the business has to do is ensure that the rules are specified so that *information elements* representing *characteristics* essential to survival are replicated in a suitable DSP at a safe and remote location.

This can be compared favorably with a distributed relational database that depends on two-phase commit strategy and thus incurs the risk of delay in direct proportion to the number of other nodes involved. The core idea of *quantization* ensures that all such risk is localized and are thus independent of any failure in other parts of the network. All information is committed locally without waiting for confirmation from other nodes. This is made possible because the only data that is replicated represents independent elements of input that can never change.

4.5 The integration problem

The way in which the *quantum data store* can ease the problems of uncoordinated application development is illustrated by the following quotation from J. Bubenko 1980 [2].

"It would permit us to implement systems gradually in a truly incremental fashion". "We could start with defining and implementing a few urgent parts of our U o D and then take other parts in any desired order".

One of the greatest factors that obstruct integration is an uneven pattern of funding across the range of application systems that need to share data. It is

common for development funding to become available in tranches associated with a specific information need or problem. It is very difficult to achieve and maintain compatibility between application systems unless such funding opportunities occur at the same time.

The technical problem that has to be overcome here is to integrate the different *conceptual schemas* of several application systems into a single evolving model in which common *characteristics* are inherited from generic entity classes. The core idea of *quantization* makes this possible by its focus on individual *characteristics* as against real-world objects as a whole.

By supplying appropriate filter criteria for each retrieval action, different application systems/users can construct different images of a real-world object from the same overall body of *information elements*. Where these different images are defined in different CBML models it is possible for an application system to be written entirely in terms of its own specialized interface while sharing data for those *characteristics* that appear in both models through inheritance.

5. Enabling technology

To fulfill the vision set out in section 2 it is necessary to produce a DSP product that will apply the core idea of *quantization* in a way that can be driven through conventional interface technology. The technology that makes this a practical proposition is a combination of a new Corporate Business Modeling Language (CBML) with specialized application of standard middleware design techniques within the DSP product.

5.1 Corporate Business Modeling Language

This new language is needed to provide a means by which the information created and used by each application system can be specified in business terms and without constraining the way in which it is structured inside the DSPs.

This gives the DSP an important freedom to ignore the conflicting notions of what constitutes an efficient data structure as perceived by different application system design teams. This in turn is a prerequisite for *quantization* and fulfillment of the vision.

CBML provides a way in which conflict between different data design thinking is circumvented by straightforward statements giving the types of information (*characteristics*) that are involved in the description of each type of real-world object with which the application system is concerned.

Nothing could be simpler. All concerns about the efficiency of data storage and distribution are delegated to the DSP. It is only with this delegated responsibility and consequent freedom to use a generic structure and powerful generic software that the DSPs can meet the

goals of automatic distribution and fine grained security can be met.

A further benefit of CBML is that an information model free of design thinking is more readily endorsed by specialists in the business domain and provides a sound platform for application designers to work on.

The reason that this cannot be achieved with entity modeling or with UML is that these languages were developed for the modeling of data and system designs. They rely on low-level technical concepts that require a high level of choice and creativity by the designer/modeler.

5.2 The Application Programming Interface (API)

The other key external interface is that through which an application programmer communicates with the DSP. The enabling technology here is consistent with established practice in which both designers and programmers work to interface classes that are mapped onto an underlying relational database.

The simple but crucial difference is that in a *quantum data store* these interface classes are generated automatically by the DSP manager package from the CBML definitions of the information with which the application system is concerned. One such interface class is generated for each type of real-world object (CBML *entity class*) and this provides a standard set of access methods for each applicable *characteristic* according to the specified CBML data format.

The key task of this API is to interpret the CBML definitions cited by the *meaning pointer* of each *information element* and map them onto an underlying generic data structure. A four-layer architecture is recommended for this software as explained below working up from the bottom layer.

Primitive Layer: The task involved here is chiefly one of mapping between the fixed generic logical internal structure of the *information base* and the data structuring capabilities of any one database engine. The purpose of this layer is to allow the complex software above to be equally valid for use with a variety of database engines.

Generic Layer: The task involved here is chiefly one of applying retrieval filters, handling the internal structures of the CBML data types, detecting inheritance through the classification system with caching to optimize access to the underlying database. Essentially it does all the hard work for the various packages within the business support layer described below.

Business Support Layer: The role of the business support layer is to create for the programmer an illusion that the data store contains instances of entity classes selected from the CBML model for the business area concerned. This is achieved through automatically generated packages of interface classes that provide

functions for retrieval and update of the characteristics defined for the entity class concerned.

Business Rules Layer: This is an optional layer that is provided as a convenient place for the inclusion of rules that control inputs in accordance with business disciplines and procedures. There is also potential for the specification of compound entity classes whose assembly would require the API to navigate between a number of related real-world objects. The motivation for such development would be the reuse of business related logic with the benefit of standardization as well as a saving in development effort, reduced testing and increased reliability.

5.3 Internal generic data structure.

This third piece of enabling technology concerns the internal logical structure for application data held inside a DSP. It is required to provide a relational or other practical data structure that can hold an *information base* with *information elements* of all CBML data types together with its *master index* and the *semantic library*.

Here it is important to recognize that this is an internal structure that does not have to be the same in every DSP provided it has generic design that is consistent with the formal definition of CBML. This flexibility is possible because all transfer of *information elements* between DSPs will conform to the XML standard for CBML structured data. Thus there is scope for competition and progressive performance tuning as well as designs tailored to favor particular CBML data types that may have particular prominence within a certain business area. These are internal design choices that have no effect on the key features of automatic distribution and fine-grained access control.

Such structures have to be generic because they must be wholly independent of all business semantics which are held as data in the *semantic library*. The structure that holds the *information base* must include distinct sub-structures capable of holding the useful content for each CBML data type. The *control wrapper* is a feature common to all these data types which begin with the simple ones of *string*, *number*, *date-time-group* but include more complex ones like *text*, *referential characteristics*, *categorizations* and *quantity*. All these are defined in the CBML Baseline which can be downloaded from the web at cbml.com.

5.4 Globally unique identifiers

Thee GUID, or globally unique identifier, has a crucial role in the quantization of information. It has no descriptive content and is designed solely for use by logical references within the data store and as a handle by application programs. Every record within a quantum data store must have a GUID. Thus there is:

A GUID for each real-world object.

A GUID for each *characteristic*.

A GUID for each *context of origin*.

A GUID for each *information element*.

For use in such logical references it is necessary to have a guarantee that no GUID can ever be reused. This is ensured by defining the GUID to comprise two components. One of these is a serial number generated automatically. The other is a centrally allocated identity for the serial number generator concerned. With 64 bits for each of these there is a potential of 2^{128} such GUIDs within a given quantum data store.

5.5 The DSP manager package

One key part of the vision is that the developers of client application systems would be relieved of all need to consider requirements for security, recovery, distribution and any form of interface with other application systems.

In order to achieve this it is necessary for control over these aspects to be placed in the hands of ordinary users. It must be possible for people to use general management skills to direct the services provided by the network of DSPs just as they would for any non-IT service or resource. The technology involved here is not difficult but it is vital that these functions be supported with user-friendly facilities as for any other business function.

When a new DSP is installed, its owner will first use DSP Manager to specify rules for acceptance of clients and further rules to specify the means, if any, by which the existence of this DSP is to be advertised. Access to the services provided by any DSP may then be obtained by anyone with knowledge of its existence and access to this same software package.

Such persons may apply for data services on behalf of an application system or as an individual seeking to use query/input facilities provided as part of the DSP Manager. Services of storage, retrieval and update of information provided to an application system would be delivered through a specially tailored API that would be generated by the DSP Manager and downloaded for incorporation into the application software as a linked library routine.

If the person or application system requiring data services is found to satisfy the rules specified for the DSP concerned then it becomes a registered client of that DSP. Such registered clients may then specify rules to govern the means, if any, by which their existence shall be advertised to other DSPs and made visible to other clients.

5.6 Data Persistence Contracts

The aim here is that the designer's of each application system should provide their users with a simple but formal way of notifying the DSP of their intentions in accordance with the requirements and capabilities of that system. This is perceived as a contract so as to

underline where the responsibilities lie. The DSP would be entirely responsible for carrying out the instructions embodied in a contract – no more and no less. The client application system will retain full responsibility for the semantic definition of all data submitted for persistent storage. Users of the application system would retain responsibility for the actual data and the context to which it is committed in each case. The DSP would have full responsibility for the safe-keeping, distribution and security of all information placed in its care but the client application systems would be responsible for the rules that govern these processes.

In this vision any registered DSP client may at any time specify any number of Data Persistence Contracts (DPCs). Each of these is activated through one or more Contract Dataset Extension (CDE) that defines certain types of information for which persistent storage and distribution services are to be provided under the contract. Further CDEs may be added to a DPC at any time. Such CDEs may be de-activated at any time and the DPC will only remain active while it has at least one active CDE.

Each Contract Dataset Extension (CDE) will be for information about one type of real-world object and will specify the types of information that make up the datasets concerned.

Each type of information (*characteristic*) cited within a CDE will be defined in a widely accessible Semantic Library. This library shall contain the sole and complete statement of meaning for each *characteristic* of each type of real-world object. The new language CBML, introduced in section 5.1, is designed expressly for this task with fully equivalent graphic, text and XML forms. The DSP Manager will allow a library of these CDEs to be built-up with display and edit facilities and ability to switch between text and graphic modes with the XML form to serve as the mark-up standard for transfer purposes.

Each valid Data Persistence Contract (DPC) will allow its owner to enter datasets in accordance with any of the associated CDEs at any time through any connection to any of the DSPs with which it is registered.

At any time while a DPC is active its owner will be able to set up or modify the range of Beneficial Client Extensions (BCEs). Each of these will extend the privileges of data access to a target beneficiary that will be either a person or application system but must already be a registered client of at least one DSP. It will not be required for the owner of the DPC to be registered with that DSP and any such coincidence will not affect the delivered service in any way.

5.7 Validation by prototyping

The relative novelty of the core idea means that one of the main questions must be “Does it work?”. The only

satisfactory way to answer this question is to do it. For this reason the British Army has undertaken a significant program of pilot software development. This has been implemented in the form of a generic data structure known as DCADM together with a form of the required API. These in turn has been tested in use by a demonstration application system based on real business needs chosen to exercise the key features of the core idea.

By these means it has been demonstrated that the storage and retrieval of *information elements* can be driven in a satisfactory way through a set of interface classes based on *categorizations* that are themselves subject to history, projection and competing versions.

The prototyping performed to date has used a popular commercially available RDBMS to handle the physical storage along the lines set out in sub-section 6.1 above. This choice was a matter of convenience and minimum effort. In principle the choice of physical storage mechanism should have no impact beyond considerations of cost, performance, reliability and ease of implementation. Possible alternatives include the use of a commercially available object database software and completely new software tailored to exploit the distinctive behavior of a *quantum data store*.

6. Impact on related data management issues

In a conventional relational database the role of the database is that of caretaker of data created and owned by an application system. The *quantization* of data alters this in a very fundamental way. The concept of database is replaced by that of a *data infrastructure* which is caretaker of a corporate resource of information about real world objects obtained from all sources.

The purpose of this section is to explore the ways in which this changed role alters some key aspects of data management and application system development.

6.1 Impact on data standardization.

One of the problems most affected by the *quantization* of data is the difficulty of establishing a viable conceptual schema with scope that extends across more than one application design authority. Both social and technical problems contribute to this difficulty.

The chief social problem is the reluctance of independent designers to commit to a corporate perception that is not tailored to their immediate needs. For this reason data tends to remain the property of one application system rather than become a truly corporate resource.

It might be possible to overcome such reluctance were it not for the technical limitations of a conceptual schema based on *entity classes* defined with fixed

relationships and attributes. The ways in which objects are classified and described in the real world are commonly too complex and too variable to be represented adequately in this way.

Quantization eases this technical problem by allowing the designers of different systems to apply their own classifications and other *characteristics* to a shared set of real-world object identities.

This shared set of real-world identities, represented by the *master index*, allows such different perceptions to inherit common *characteristics* from a partial but expandable set of agreed definitions. This means that progressive standardization can be exploited by each application system as and when opportunity presents.

6.2 Impact on data distribution and integrity.

The effects of *quantization* on the integrity of data distribution are potentially highly desirable. The beneficial effect stems chiefly from the invariant nature of the *information elements*.

Since each *information element* represents a single state of a single *characteristic* of a single real-world object at a single point in time it can never change. This means that even though copies may be widely distributed there are no changes to be synchronized. This invariant nature is of major importance within a network that is subject to frequent modification or vulnerable to attack because all dependence on two-phase commit locking and hence all risk of deadlock due to communication failure is removed.

This simplification derives from the fundamental shift in aim from “a database that represents the changing state of a system” to a logical data store that represents “a superposition of all known states of all *characteristics* of all real-world objects within a declared scope”.

Each *locality database* behaves just like one that is fully independent except that it has the benefit that local input is supplemented by contract driven feed of other relevant inputs that happen to have been entered at different *locality databases*.

6.3 Impact on history and archiving.

The problem considered here is that there is no reliable way of predicting requirements for historical analysis. The effect of *quantization* is to preserve everything forever within the logical space of a *quantum data store*. This is a direct result of the core idea and is achieved simply through the generation of a *GUID* for every *information element*. This means that every value supplied from any source at any time for any *characteristic* of any real-world object has its own *GUID*.

The practical consideration of how many of these *information elements* are represented by a physical image within any *locality database* is a separate matter

that can be managed to reflect the capacity of that database and the priorities of its users. Archiving thus becomes a matter of designing the network to include specialized archive databases that behave just like any other *locality database* but have no local input and are designed with very high capacity media suitable for long-term storage and high speed searching.

6.4 Impact on access control.

The problem considered here is how to control the distribution of *information elements* so that each user has local access to all relevant information on a need-to-know basis. The key to this is that part of the control wrapper called the “context of origin pointer”. This is set when the *information element* is created and has the form of a reference to the *GUID* of a particular sort of conceptual object known as an information context.

This concept of access control requires an information context to be created for each user role. This user role is then empowered to create sub-contexts and to allocate access permissions at will so that other selected user roles are given permission to read or write *information elements* having that sub-context as their “context of origin”.

Each user role is required to direct each commit operation to one information context selected from those that it has created and those owned by other user roles for which it has been granted write permission. All retrieval operations on behalf of that user role are limited to *information elements* that have their origin in one of the information contexts for which that user role has current read or write access permission.

The effect of *quantization* is to enable each user to control access to any information they create and to do this selectively from time to time at any desired level of granularity. Because control is exercised at the level of *information elements* it is possible for a real-world object to have descriptive detail emanating from many different contexts while each user sees only those that are appropriate.

6.5 Impact on resiliency.

The problem considered here is the need to ensure that change, loss or damage to any *locality database* or the communications between them cannot affect the integrity of the data that remains. *Quantization* greatly increases the inherent resilience of the data store. This is chiefly because each *locality database* is self-sufficient in that it contains all *information elements* that its local users have a right and need to access.

This means that the only possible adverse effects of events elsewhere in the network are to deprive it of inputs that might otherwise have been copied to it. There is the further mitigating effect that there is much scope for repair by using *information elements* from surviving *locality databases* to restock a replacement

locality database or to equip another *locality database* to support additional users.

6.6 Impact on overall performance

The effects of *quantization* on system performance are best measured in terms of the resource cost per task. Work on the creation of sizing parameters suggests that significant impact may be expected in two areas.

The first of these is the number of bytes needed to hold any given piece of information. Analysis of demonstration data suggests that the number of bytes required may be expected to increase by a factor of between three and eight. This is governed chiefly by the relative size of the useful content and control wrapper for the average *information element*.

The second aspect is the cpu usage taken-up in the overall process of storage and retrieval as viewed by the user. Experience with a small-scale demonstration package has shown good response. Measurements have also been made under test-bed conditions. The design of the current implementation divides the process into two distinct parts.

First there are the new tasks consisting mainly of re-expression of *entity class* based external schema views into the generic and *quantized* form in which the data is held. Measurements taken to date suggest that the average cpu cost here is no more than ten per cent of that consumed by the underlying relational database.

The other part of the process is the work undertaken within the underlying database server. This cost is largely dependent on searching strategies and the favorable results on limited data volumes have little significance. It is clear that for large volumes of data it will be necessary to exploit some of the extensive scope for optimization. In particular the classification system contains many arrays that are frequently traversed but rarely changed. These are prime candidates for caching. A brief study has indicated that object database technology may prove suitable in this role.

6.7 Impact on information system development

It is not envisioned that the user or application program will address the data store directly. Instead it will engage in a dialogue with the data infrastructure as a whole. The nature of this dialogue is that between consumer and provider of data storage services.

An application program connected to any *locality database* can create, retrieve and modify the recorded *characteristics* of any real world object. It does this by invoking methods provided in a range of interface classes. There is one such interface class for each *entity class* defined in the conceptual schema and this has methods for the retrieval and update of all associated *characteristics*. It is important to appreciate that functions provided to 'modify' or 'update' *characteristics* are implemented so as to create new

information elements and to make other ones obsolete as appropriate.

The overall approach to stored data begins with selection of real world objects. This is followed by retrieval/update of individual *characteristics* and/or navigation to related real-world objects which are then addressed in the same way.

The main difference from the typical relational database application is that the application is dealing with *characteristics* individually rather than with the object as a whole. The retrieval and update methods for each *characteristic* are invoked with a parameter containing the *GUID* of the real world object concerned.

To the application programmer this *GUID* serves as a handle for the real world object while it is being addressed. This handle is obtained either by selection from the results of a data store search or from a referential *characteristic* that is already to hand.

It is the general intention that the fragmentation of stored data into *information elements* should be hidden from the application developers and regarded as just something internal that is necessary to enable the sharing and replication behind the scenes.

7. Conclusion

This is a white paper that stems originally from the pressing and unsatisfied needs of a major user of information technology. It attempts to show that the vision set out in section 2 can be achieved and that the core idea of *quantization* set out in section 3 does provide the key to the sharing of live application data across the internet.

Section 5 gives an outline description of major aspects of the enabling technology that has been developed by the Army. It is hoped that this outline taken together with the arguments in section 4 and the assessment of impact in section 6 will provide enough evidence to show that the work merits wider attention within the industry and in the academic world.

Serious take-up of these ideas requires investment that depends chiefly on the degree to which the industry can be persuaded that this new technology is sufficiently practical for the vision to be achieved. It also needs to be persuaded that the envisioned Data Service Points will prove to be an attractive investment for user organizations and can be shown to pay for itself through easement of the costly problems of data integration, management and security that are faced by many IT managers.

Exploration of this potential market is currently at a very early stage. The aim of this paper is to stimulate interest and attract constructive comment on the whole idea. All comments will be much appreciated by the author at harryellis@compuserve.com

References

1. Bubenko, J. A., 1983, Information and Data Modeling – State of the Art and Research Directions. In Kangassalo, H (ed), *Proceeding of the Second Scandinavian Research seminar on Information Modeling and Data Base Management*. University of Tampere, Tampere, Finland, 9-28.
2. Bubenko, J. A., 1980, Information Modeling in the Context of System Development. In Lavington, H.A. (ed), *Proceedings of the IFIP World Congress*. North-Holland, Amsterdam, 395-411.
3. Langefors, B., 1973, *Theoretical Analysis of Information Systems*. Auerbach, Philadelphia, PA. Chapter 6.

Using the Quantum Data Model to Develop Shareable Definitions

Harry Ellis

British Army

Army Data Services, Faraday Building, Blandford Camp
BLANDFORD FORUM, Dorset, DT11 8RE,
UK

harryellis@compuserve.com

Abstract. This paper contains an outline of a new form of data model with particular focus on the way real-world objects are classified. Concepts such as categorization schemes and designation schemes that are widely used in the real world are made available for conceptual modeling in a new language called CBML. Case material is used to show how this new language can help to reconcile different perceptions of information. In particular it is shown how this can lead to an enterprise model with formal links to those of specific business areas and information systems.

1 Introduction

This paper introduces a new form of conceptual modeling that has emerged from practical experience within the British Army. This development is a result of extensive research by the author and colleagues within Army Data Services (ADS). This research was undertaken while striving to create a large conceptual schema with the clarity and rigor needed to support the sharing of information across this large and complex enterprise.

1.1 Technical Development Driven By Necessity

Data sharing is an imperative for the British Army. It has a massive legacy of disparate systems, information needs that are changing all the time and a goal of “fully interoperable information systems by 2010”. It was found that some of the difficulties faced were due to deficiencies in the languages available for conceptual modeling. For this reason ADS decided that it had to take a fresh and critical look at the concepts, methods and tools that are currently available for conceptual modeling.

1.2 What is the Quantum Data Model?

The *quantum data model* is a set of concepts for use in defining the structure and meaning of information. It has been devised by ADS as a necessary step towards the Army’s interoperability goal. It differs from the currently dominant *relational model* by a shift of focus away from tables that represent types of real-world object and toward the various types of information that are used for their description.

This new set of concepts can be seen as a synthesis of ER and the data aspects of UML together with some key innovations. The roots of this radical development lie in difficulties arising from the limited forms of generalization that underpin all the established languages for conceptual modeling. It was found that these difficulties could only be overcome with the addition of certain new concepts. The name *quantum data model* was chosen because these new concepts describe an enterprise in terms of "the superposition of all states of various types of information about real-world objects of various types". This echoes the general idea of "the quantum description of a system".

1.3. What Is New About It?

All widely used languages for conceptual modeling of information are based on the notion that each real-world object can be represented as an instance of a single entity class. It is further assumed that this entity class should determine the nature of the attributes and relationships by which real-world objects of that type are described. In ADS we found that real-world objects are not that simple. The currently dominant approach to conceptual modeling leads to attributes and relationships that are tainted with thinking that is more in tune with data design than with the way in which information is perceived in the real world.

The *quantum data model* contains three important technical ideas that allow the most complex information to be defined in a consistent way by simple unambiguous statements. We found that dialogue with end users works best when focused on discrete elements of information that are self-contained units of change. Some of these are simple and could be represented by an attribute or foreign key. Others are quite complex and would need a whole table if normalization rules were strictly applied.

We found that real-world objects can be categorized in many different ways and that in many cases this categorization may vary over time. We also found that the many entity classes involved in a substantial enterprise need to be grouped into sets within which certain rules apply. Some of these entity class sets serve as *designation schemes* used to classify real-world objects when they first become of interest. Other entity sets known as *categorization schemes* provide a rational framework for subsequent further categorization.

1.4 Why Was This Development Necessary?

The development of innovative information technology is not a core function for either the British Army or its data services organization. For this reason every endeavor was made to meet the Army's need for an enterprise level conceptual schema with well established modeling techniques. The prime candidates for this task were ER (Barker) and UML but both of these proved unsuitable for a conceptual schema that must be able to grow and adapt with changing business needs. It was only when it became apparent that these languages were causing more problems than they solved that we were forced into the fundamental thinking that led to the ideas of the *quantum data model*.

Having developed this new body of concepts it was necessary for them to be formalized into a language with support tools and training. The name "Corporate

Business Modelling Language” (CBML) is derived from its *corporate* (enterprise) viewpoint and focus on *business* (real-world) objects. For the remainder of this paper, discussion of the general principles of the quantum data model is illustrated through the specific terminology and graphical notation of CBML. Full definition of this language can be obtained from the website: www.CBML.co.uk [1].

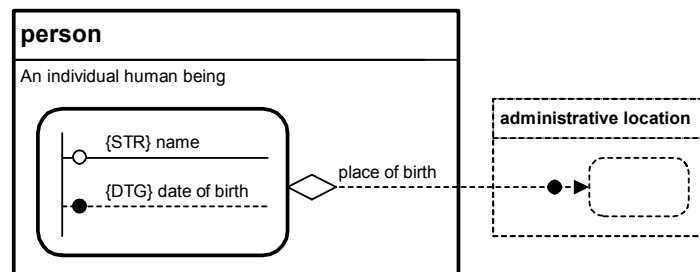
2 Key Features of CBML

The key difference between CBML and the established languages for conceptual modeling is a shift of focus from entity classes on to individual *characteristics*. These characteristics are elementary units of information that have meaning on their own but cannot be split in any way without destroying that meaning.

The effect of this shift of focus is profound. It enables us to say categorically that every entity class represents a type of real-world object and never represents a type of information. With this new focus it becomes possible for the modeler to formally distinguish types of information from the types of real-world object that the information describes, i.e. to distinguish between the content and the subject.

2.1 Characteristics

Information is represented in CBML by *characteristics* while subjects are represented by entity classes. The function of the conceptual schema is now easy to explain. It says which types of content (characteristics) are used to describe which types of real-world object (entity classes).



Each instance of **person** is an individual human being described by:

"name" (mandatory, variable, character string)

"date of birth" (optional, fixed, date-time)

"place of birth" (optional, fixed, ref to single instance of administrative location)

Fig. 1. Example of CBML graphical notation

This first glimpse of CBML in Figure 1. illustrates the basic style of its graphical notation. It shows that the entity classes can be modeled individually and that reference can be made to another entity class without knowing anything about it beyond its name. The most striking visual feature of CBML is its use of a double box. The outer box (deliberately reminiscent of a UML class) represents the entity class as

such, e.g. the concept of “person”. The inner box (deliberately reminiscent of an ER (Barker) entity soft box) represents the instances, e.g. unspecified individual persons.

The reason for drawing this distinction is to aid verbalization and diagram readability. The model expresses rules. Some of these rules are expressed as a definition of the class as such. A good example of this is the generalization concept in UML. Other types of rule are associated with the class but are expressed as applicable not to the class as such but to its members (instances). A good example of this is the verbalization of relationships in ER (Barker). The double box notation enables a CBML modeler to attach class definitions to the outer ‘class’ box while rules that apply to its members are attached to or placed inside the inner box. We have found that this makes it easier for user representatives to read the models and appreciate their full implications. In visual terms this gives the best of both worlds - an ER model embedded inside a UML class diagram.

2.2 History, Projection and Competing Versions

One of the key problems faced in our modeling was to decide which characteristics needed history, projection and/or allowance for competing versions. It was found impossible to say for any characteristic that these would never be required. This threatened to make the model both complex and subjective. To avoid this we decided that the model should incorporate an assumption that the implementation would provide for history, projection and competing versions for all characteristics. This underlying provision could then be invoked as operational circumstances required. This is a big assumption but is arguably sound since the usual presumption of ‘latest value only’ is just a limiting case and would be covered anyway.

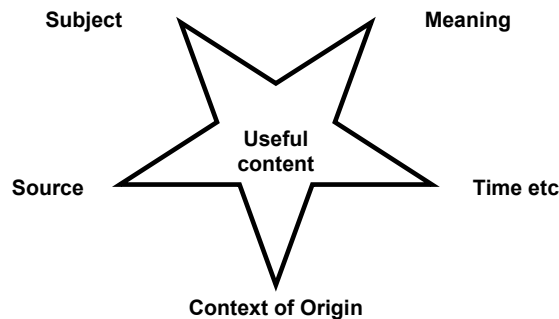


Fig. 2. Information Elements

Figure 2 illustrates the core idea of quantization. It is this principle that has enabled ADS to provide an implementation vehicle that deals with history, projection and competing versions in a comprehensive and thus economical way. Each element of information is stored such that its *useful content* is forever associated with a control wrapper that serves as a sort of passport that enables it to reside, move and be copied anywhere in cyberspace. This ‘passport’ contains five vital pieces of control information.

Subject. This identifies the real-world object that is described. It does this by citing the globally unique identifier of the real-world object concerned.

Meaning. This identifies the relevant characteristic as defined in the conceptual schema.

Source. This is a pointer that identifies the person, organization or information process that created the useful content.

Context of Origin. This identifies the operational situation, activity, geographic location etc to which the useful content is relevant.

Time Etc.. This refers to a set of qualifiers that, together with a globally unique identifier, serve to distinguish different versions from the same source with the same context of origin such as a locus of positions for a moving vehicle.

2.3 Complex Characteristics

It was said earlier that all characteristics for any given real-world object must each have their own independent life history comprising a set of *information elements*. This means that the ‘useful content’ must have a structure that is appropriate for the characteristic concerned. Some characteristics like ‘name’ are easy. The ‘useful content’ is a single character string. It is not always that simple. If the self-contained unit of information represented by a characteristic is more complex than any of the standard data types of ‘string’, ‘number’ etc. then an appropriate abstract data type must be created.

2.4 Categorisation

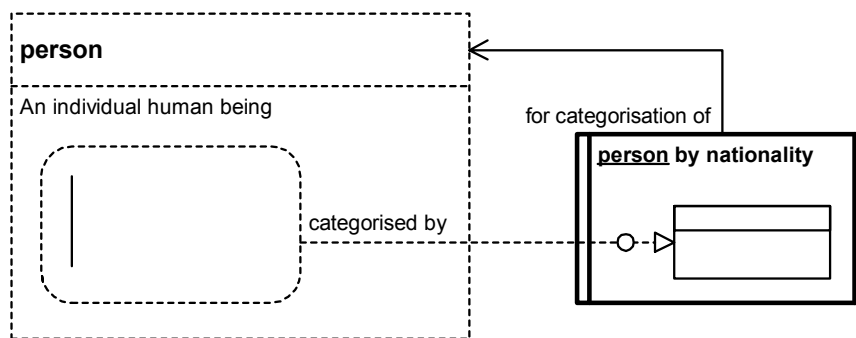
The new type of *characteristic* known as *categorization* is needed to show clearly where information about a real-world object implies additional typing with associated inheritance. This type of characteristic is defined as a reference to a set of *abstract entity classes* representing a choice of categories. *Characteristics* defined for these abstract entity classes are inherited through the *categorization* but only while it is in force.

This concept of categorization is of great importance. What we are talking about here is usually modeled as either an attribute from an enumerated domain or a reference to an entity class that represents a set of type definitions. We have found that both of these concepts are deeply flawed when used in a conceptual schema. It is recognized that this is a non-trivial issue. Work is in hand to address the issue of generalization in the conceptual schema in a full research paper. The point being made here is that the concept of categorization is well understood and widely applied in the real world and for this reason must be recognized formally within any language used for the expression of a conceptual schema. By ‘well understood’ is meant that people in general are inclined to presume that if “A is a B” then everything known about B must apply to A. This much is intuitive and may be assumed by user representatives wherever any detail in a conceptual schema can be read or interpreted as a statement in the form “A is a B”.

2.5 Entity Class Sets

In languages currently used for conceptual modeling the only way in which entity classes can be grouped is as sub-types of a more generic entity class. The problem with this is that across an enterprise there may be many such sub-types that have little to do with each other. Suppose we have four sub-types of 'vehicle': 'land vehicle', 'sea vehicle', 'military vehicle' and 'civilian vehicle'. In the real world these are recognized as falling into two groups: 'vehicle by mobility' and 'vehicle by military/civilian status'. To classify an instance of vehicle it is necessary to pick one type from each group. Groups like this are represented in CBML by sets of entity classes known as *designation schemes* or *categorization schemes* according to the purpose for which they are defined.

The following diagram shows how a *categorization scheme* "person by nationality" would be expressed in CBML. Note that two relationships are involved here, each being a distinct form of generalization. First there is the statement that the *categorization scheme* "person by nationality" is *for categorisation of* persons. This says formally that members of this scheme such as 'British person', 'Japanese person' etc. are all types of person and must not be used to categorize any real-world object unless it is already known to be a person of sorts.



"person by nationality" is a categorization scheme whose members may be cited as categorisation of instances of **person**"

"Each instance of **person** may be categorised by reference to one variable member of the categorization scheme **person by nationality**"

Fig. 3. Categorization Scheme

The second statement represented by the broken line labeled 'categorized by' is a rule for generalization of instances of person. In the case illustrated here the rule says that categorization of persons by nationality is optional and variable but allows only one category at a time. This would exclude dual nationality and thus represents a rather narrow view. In CBML it would be easy to alter this to indicate the possibility of multiple nationality by using a double-headed arrow.

2.6 Designation Scheme

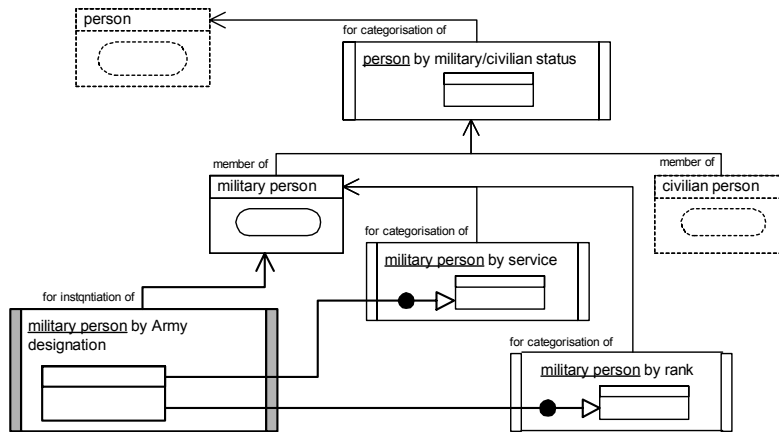


Fig. 4. Designation Scheme

This example shows a designation scheme called “military person by Army designation” whose members are concrete entity classes. Such schemes are found wherever a business area has devised its own set of type designations each of which incorporates a distinctive name and a particular combination of further categorizations. In the case illustrated, the Army is shown as having its own set of type designations for military person where each represents a specific combination of service and rank e.g. ‘army major’ and ‘navy captain’. When a real-world object is created as an instance of the type designation ‘army major’ it is immediately and permanently defined as inheriting applicability of the characteristics defined for the four entity classes: ‘person’, ‘military person’, ‘army person’ and ‘major person’.

3 Comparison With Other languages

3.1 Why Not Use ER?

ER modeling is designed to identify “the things of importance in an organisation, the properties of those things and how they are related to one another” [2]. It has simplistic inheritance structures and it relies on the modeler to choose between alternative modeling constructs. The only reliable definition of the entity concept is as something that can usefully be represented by a data table. For this reason the language is suited more to ‘logical’ than ‘conceptual’ modeling. The simplicity of the language makes it easy to use but very difficult to use well. The creation of entity models that can be read accurately by persons other than their creators demands a combination of high skill and inventiveness with rigorous control of style. This combination is largely unobtainable in the number of modelers needed by a large enterprise.

3.2 Why Not Use UML ?

“UML is a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software intensive system” [3]. This categorical statement from the originators of UML provides a fair indication of the unsuitability of UML as a language for the construction of a conceptual schema. One of its great strengths is the inheritance structure but this is more rigid than those used in the real world. Another strength lies in the process methods but these are of little, if any, relevance to the aims of a conceptual schema. The problem here is that the real world of an enterprise does not behave like a single rule-based machine. It is more like a federation of active resources intent on pursuing individual goals through collaboration. This cannot be described in UML. At a more superficial level it is found that both verbal and graphic representation of associations is poor. There is also the problem that data hiding which is great as an implementation device is inappropriate for the definition of information that needs to be shared.

3.3 Why Not Use ORM ?

ORM “is a semantic modeling approach that views the world simply in terms of objects playing roles (parts in relationships)” [4]. ORM is aimed directly at the conceptual schema and its verbalization is a strong feature. The graphics are rather cumbersome and play a relatively minor role in the endorsement by user representatives. The main problem is that the verbalized facts can lead both modeler and user to read far more into a model than is formally defined. This derives from the great reliance on its similarity with natural language and wide choice of expression. It also suffers from the same limited form of generalization as UML.

3.4 Why Is CBML Better ?

The above criticisms of ER, UML and ORM cannot be justified here for want of space. The aim here is not to condemn the use of these languages for conceptual modeling but to introduce CBML and explain how and why it is different. The main strength of CBML derives from its being designed to define “types of real world object and the types of information by which they are described”. It is for this reason that it has formal ways of expressing multi-way sub-typing and transient inheritance as used in the real world. The clear distinction between type and instance means that the need for creative modeling can be eliminated. This means that the modeler can concentrate on the analysis of information needs and their expression in a standard way that all-comers can read with the same understanding. Another major strength is the ability to define an entity class to include references to other entity classes whose own characteristics are as yet undefined. This means that models can grow and adapt without rework.

4 Data Standardisation in a Large Enterprise

The main driver for the development of CBML was interoperability of information systems. This requires standardization at best and reliable two-way mapping at worst. It is hoped with some justification that CBML will facilitate the creation of enterprise definitions that are truly independent of implementation choices. These can then be

implemented in various ways in the certain knowledge that the information represented is meant to be the same and is properly defined in the conceptual schema.

To achieve this the language must be supported by a method that can cope with a pattern of application development in which progress is made at different rates at different times in different parts of the enterprise. This means that legacy data is always important. The method developed around CBML is based on an awareness that a conceptual model must satisfy two conflicting aims. To be properly endorsed it must relate directly to user perceptions while for implementation it needs technical precision. In CBML this gulf is bridged by formal recognition of five distinct levels of model. These levels provide a disciplined framework within which different perceptions of information can be reconciled. The five levels of information model are set out below.

4.1 Legacy Models

Legacy models are at their most important in the early stages of establishing a conceptual schema for the enterprise. There are two distinct levels as follows.

Actual Legacy Models (Level A). Models at this level are a faithful representation of actual data as it exists. The relational data model is a form commonly found at this level.

Baseline legacy Models (Level B). Each of these is an adaptation of one *actual legacy model* in which each possible interpretation of each type of information has been given its own distinct representation and name.

4.2 CBML Models

CBML models are genuine conceptual models that may differ in purpose, scope and quality as follows.

Candidate CBML Models (Level C). These are models defined in CBML that express the information needs of a specific business area in terms that reflect current user perceptions.

Developed CBML Models (Level D). Each of these is a developed version of a specific *candidate CBML model* that has been refined in accordance with CBML best modeling practice but without consideration of any needs that lie outside the scope of the business area concerned.

Enterprise CBML Models (Level E). each of these covers the information needs that are common across a complete enterprise. As such it is a synthesis of all relevant *developed CBML models*.

4.3 Comparison of Levels

All CBML models are required to conform fully to the definitions and graphical notation. However, for *candidate CBML* models the overriding consideration is the

ease with which the people who use the information can understand, review and endorse the model. It follows that the definitions of characteristics should be simple and direct even though this may fall short of best modeling practice in some respects. It is in the progress to level D that the precision needed for implementation is applied. This is a prerequisite for any attempt to reconcile the conflicting needs of different business areas in order to create agreed definitions for inclusion at level E.

5 Case Study

The following illustrations are taken from actual modeling experience on an aspect of army information which is of concern to two distinct business areas. One of these is the central personnel function. The other area covers the individual fighting and support units where the immediate information needs of these units are heavily influenced by a major legacy system known as UNICOM. The aim of this case study is to show how CBML helps with the reconciliation of different perceptions of common information.

5.1 Development of Conceptual Schema For a Business Area

The model shown in Figure 5 shows the way in which the concepts of “person” and “military person” were initially perceived by the business area for personnel. This is a *candidate model* (level C).

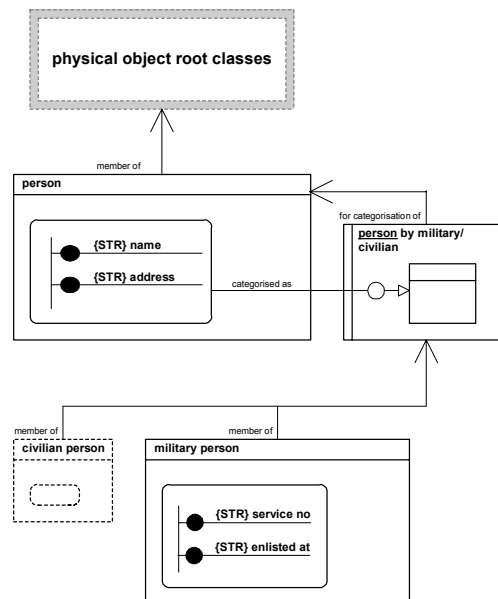


Fig. 5. Candidate Model – Personnel Area

This model says that each person must have fixed string characteristics for ‘name’ and ‘address’ and a variable categorization as either military or civilian. Variability is indicated by the blobs which are solid for ‘fixed’ and hollow for ‘variable’. The single

hollow arrowhead means ‘categorization by reference to only one member category at a time.

The concept of ‘military person’ is shown as an abstract entity class and member of the categorization scheme “person by military/civilian”. This states formally that all instances of ‘military person’ inherit, from the entity class ‘person’, applicability of the characteristics ‘name’ and ‘address’. In addition to this inheritance there are two further string characteristics called ‘service number’ and ‘enlisted at’.

In the developed version shown in Figure 6 the representation of the characteristic ‘enlisted at’ has been changed from a string to a referential characteristic that is mandatory, fixed and refers to just one instance of a new entity class called ‘recruiting office’. This new entity class is shown with broken outline to indicate that it is not fully defined on this diagram and may not yet be defined anywhere.

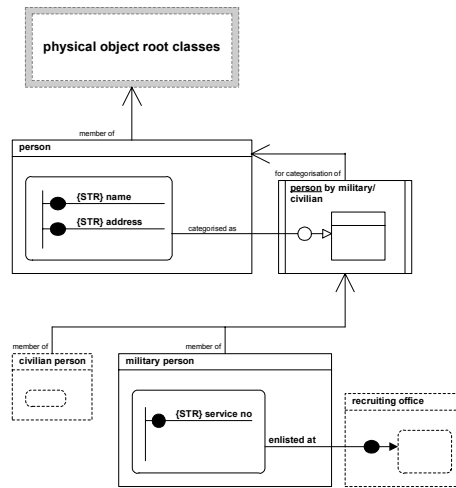


Fig. 6. Developed Model – Personnel Area

5.2 Development of Conceptual Schema For a Legacy System

Figure 7 shows a candidate model for part of the UNICOM system. This shows three entity classes each based on a table within the existing database.

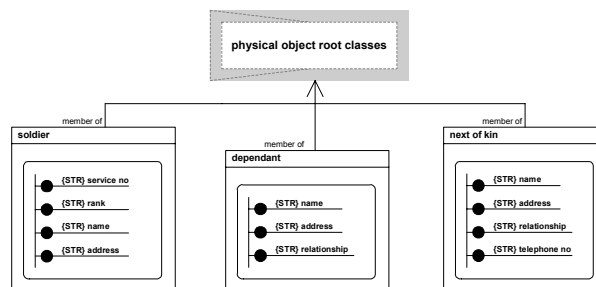


Fig. 7. Candidate Model – UNICOM System

It is immediately evident from this diagram that this model is somewhat naïve for a conceptual schema since all of the characteristics are mandatory, fixed character strings. The improved version shown in figure 8 reflects a recognition that the concepts of ‘soldier’, ‘dependent’ and ‘next of kin’ are all roles of ‘person’.

Further development based on application of rules for good modelling practice has the characteristic ‘rank’ changed into a mandatory but variable categorization reference to any one member of a new *categorization scheme* called ‘army person by rank’. This is illustrated in figure 8.

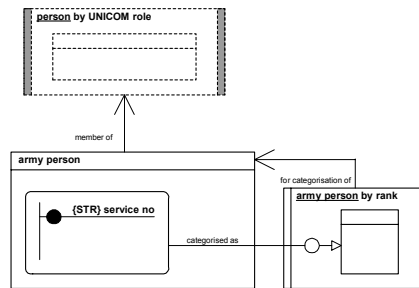


Fig. 8. Developed Model – UNICOM System

5.2 Enterprise Model

When the developed models of the Personnel Area and the UNICOM system are compared it is clear that the more generic concepts of person with ‘name’, ‘address’ and categorization by military/civilian are shared by both. Figure 9 shows how this is reflected in the enterprise model

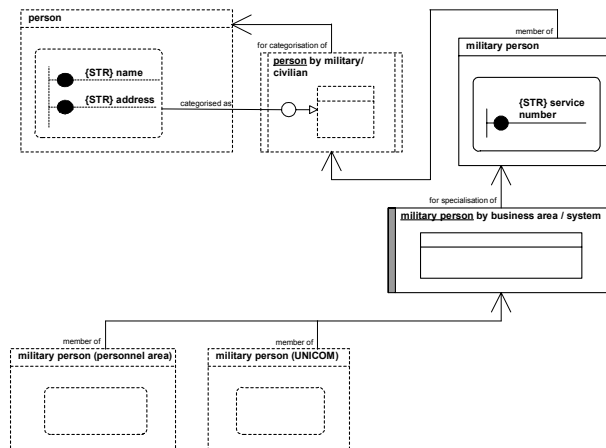


Fig. 9. Enterprise Model

A *designation scheme* is introduced with the name “military person by business area / system”. This has member entity classes called ‘military person (personnel area)’ and ‘military person (UNICOM)’ that are fully defined only within the developed

models for the respective business area / system. We now have a situation in which each business area / system has its own specialized version of military person that inherits from the enterprise model the common concept of military person with the characteristic 'service number' and further inheritance of the concept of person with its characteristics 'name' and 'address'.

6 Conclusion

This brief introduction to CBML has attempted to explain why one major enterprise has seen fit to question and improve on the established languages for conceptual modeling. The new concepts described here have emerged from a determination to breathe life into the idea of a conceptual schema in the context of a large and complex enterprise with a growing need for the sharing of information.

While the thinking behind this development owes much to the pressures faced by the British Army the impact of global communications is applying similar pressures in other fields. It is hoped that the wider research community will consider these ideas to evaluate their technical merit and potential for wider application.

References

1. Army Data Services: Corporate Business Modelling Language (CBML) Baseline (Version 2.0), Available from <http://www.cbml.co.uk> published by Army Data Services, Faraday Building, Blandford Camp, Blandford Forum, Dorset, DT11 8RE, UK (2001).
2. Barker, R.: Case*Method Entity Relationship Modeling, Addison-Wesley Publishing Company (1990).
3. Rumbaugh, J., Jacobson, I. and Booch, G., The Unified Modeling Language Reference Manual, Addison-Wesley (1999).
4. Halpin, T.: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design, Morgan Kaufmann Publishers (2001).

A SOLUTION TO THE PROBLEMS OF SHARING DYNAMIC INFORMATION
BETWEEN SYSTEMS THAT ARE SUBJECT TO
INDEPENDENT AND ASYNCHRONOUS DEVELOPMENT.

Abstract

Contents

Introduction

The Problems Addressed

The Solution Proposed

Principles With Associated Technical Justification

Persistent Storage Of Object Data

A Common Object Pool.

Replication.

Archiving.

Scalability And The Generic Schema

An Example – The DCADM

Interfacing With Non-Conformant Data

Effect On The Task Of Application Development

Management Issues And Other General Observations

Conclusion

Harry Ellis, FBCS

A Solution To The Problems Of Sharing Dynamic Information Between Systems That Are Subject To Independent And Asynchronous Development.

HARRY C. ELLIS

Babelease Ltd. and Army CIS Authority (Army Data Services from 1st April 1999), Faraday Building, Blandford Camp, BLANDFORD FORUM, Dorset, DT11 8RE.

E-Mail harryellis@acm.org

Any attempt to share dynamic information between systems that are subject to independent and asynchronous development has to overcome the near impossibility of obtaining agreement on objects, obtaining funds for parallel development, devising mappings between different data representations and anticipating the pattern of growth in user expectations. This paper describes a solution based on ten principles for the design of shareable persistent object stores. These principles ensure that each party can buy into each extension as and when their financial priorities allow. They also remove the mapping problem by use of a stable generic physical data structure. Application of these principles is illustrated by a brief description of the Defence Command and Army Data Model (DCADM®) which has been adopted as the Army's Data Exchange Standard and preferred design for command and control databases. This comprises a master index and seven abstract data types for descriptive detail all designed in full accordance with the ten principles.

Introduction

1. The purpose of this paper is to express the technical underpinning for a major body of work that has been undertaken by the Army with others¹ in pursuit of their goal of “a single and fully interoperable Army CIS for use in peace and war, on and off the battlefield, by the year 2010”. What began in 1993 as an exercise in corporate data modelling has developed into a practical framework for the development of application systems that have interoperability designed-in. This framework includes a generic physical database structure together with an object oriented API that can be extended progressively as user needs are identified.

The Problems Addressed

2. Any attempt to share dynamic information between systems that are subject to independent and asynchronous development has to overcome a number of intractable problems. Such sharing is, of course, quite different and much more complex than the simple passing of individual versions of documents etc in response to specific triggers. The problems addressed in this paper include the near impossibility of:
 - a. Obtaining agreement on definition of the real-world objects to be addressed.
 - b. Obtaining funds for parallel development to keep systems in step as their functionality is enhanced.
 - c. Devising mappings between different data representations of the same real-world objects.
 - d. Anticipating the pattern of growth in user expectations.
3. This list is far from complete but is enough to account for the notable lack of any sharing of navigable² structured data beyond the confines of a single integrated system development project other than through the exchange of messages with predefined format. It is outside the scope of this paper to elaborate on these problems which are well understood by current practitioners in the field of defence command and control system development.
4. These problems are of particular concern to the Army where the need to share information with a range of other bodies is a matter of life and death. UK forces are now required to operate at short notice in close collaboration with a wide range of UK, NATO and other and international partners to perform a range of roles in circumstances that are impossible to predict in detail.

¹ This work has been led by the Army CIS Authority (Army Data Services from 1st April 1999) in collaboration with data management teams from the Army, Royal Navy and RAF. It also incorporates significant developments made by an international team led by the NATO Allied Data Systems Interoperability Agency during 1996.

² The term ‘navigable’ is used here for data in which duplication within any one database instance is avoided by the use of cross-references like the foreign keys in a relational database.

5. As in any other business, the Army is increasingly dependent on computer systems in its day to day work. Effective collaboration demands that its many computer systems should be able to share data with those of its partners. It is for this reason that substantial effort has been invested in the search for a solution to the above problems. The solution outlined in this paper has emerged from this work.

The Solution Proposed

6. This paper outlines a practical approach to system development that solves the above problems by:
 - a. Basing all persistent storage of data on a single physical format that remains stable through progressive extension of the range of real-world objects represented.
 - b. Allowing each project to buy-in to the growing pool of common information at its own pace.
 - c. Future proofing the design by using powerful structures that remain viable through every stage of development from simple beginnings through to the full complexity of the real-world object.
7. The solution that has emerged is dependent upon a number of key ideas. These are summarised as key principles in Table 1. It is argued in this paper that adherence to these principles is the key to overcoming the problems listed above. Not one of these ideas is new but in combination they have provoked the sort of widespread resistance that is usually reserved for a radical departure from currently established wisdom. It is for this reason that we are reluctantly forced to accept that our essentially pragmatic endeavours have led to what amounts to a new paradigm for system development.

Principles With Associated Technical Justification

8. The principles listed in Table 1 contribute in various ways to an easing of the intractable problems listed in paragraph 2 above. They do this by avoiding conflict between projects and simplifying the technical issues involved as explained in three groups below:
 - a. **Support For Many Functional Viewpoints**. The three principles in this group combine to create a situation in which **agreement on the definition of objects** and their behaviour is required only from those system developers who have an immediate and active interest in the objects concerned (ref 2.a above.). Extension of behaviour to meet new requirements does not affect existing or potential users that have no need or are currently unable to handle the new functionality. Each party buys into each extension as and when their financial priorities allow. This means that it is **not necessary to secure funding for parallel development to keep all systems in step as their functionality is enhanced**. (Problem 2.b above). For most projects a small investment of effort should be enough to secure this buy-in option by monitoring the work on object design that is undertaken by lead players and injecting an awareness of their own future needs. The three principles and associated technical justifications are:
 - (1) **Principle No 1 - Information to be shared must be defined as a single, coherent set of object classes with agreed behaviour but no visible data**. The only reliable way to express the full semantics of data is by defining the types of behaviour that may have contributed to its present state. All data must be hidden as the only way to prevent corruption through non-standard behaviour.
 - (2) **Principle No 2 - It must be possible for any real-world object to be classified from many different functional viewpoints at the same time with provision for history, projection and competing versions from different sources**. The primary goal here is to allow the same data to be used by a range of applications in support of their different functional viewpoints. Unless the underlying data structure allows classification from many different functional viewpoints each of these will spawn their own objects with inevitable duplication of some descriptive detail. In such a situation some reliable means must be found to identify the overlap and keep the different versions in line. The simplest way to avoid such duplication is to have a single master index entry with multiple classifications. The need for history, projection and competing versions arises because classification is itself a form of descriptive detail. It cannot be restricted to one authoritative source and is subject to change over time.
 - (3) **Principle No 3 - It must be possible for a real-world object to have behaviour that depends on the functional viewpoint from which it is addressed**. Descriptive detail about a real-world object is independent of the functional viewpoint from which it is addressed but this is not true for behaviour that is provided expressly to

meet the needs of particular applications/users. Classification of real-world objects by functional viewpoint provides a means by which the applicability of particular behaviour can be controlled.

- b. **Conditions For A Comprehensive Standard For Data Exchange.** The five principles in this group work together to allow a stable physical schema which can support a number of functional viewpoints at the same time. This removes any justification for having more than one form of data representation of any given real-world object. This in turn **removes the mapping problem** that makes the replication of navigable data so difficult (ref. problem 2.c above). Even where non-conformant representations remain the existence of a comprehensive and stable physical schema provides a durable foundation for data exchange standards. This means that any application can discharge all its obligations for information sharing through one interface instead of an ever-growing range of bilateral ones. The five principles and associated technical justifications are:
- (1) **Principle No 4 - The data representation of all real-world objects must be at the atomic level. By this is meant that every atomic fragment³ of descriptive detail that can be subject to independent change must have a distinct and independent existence.** If this principle were not obeyed it would be possible for some fragments to be stored in groups with a single shared identity. If this were allowed there would be no way for the database to show the different circumstances (source, effective date-time etc) under which the values of individual fragments are set. Loss of such control information is unacceptable in an environment where responsibility for setting such values may be shared between several projects. There is also the problem that any change in the validity status of one fragment would affect every combination of which it forms a part. Measures to deal with such issues are likely to involve some data duplication with consequent control problems and waste of storage and communications capacity.
 - (2) **Principle No 5 - Every atomic fragment of descriptive detail must remain unchanged from the moment of its creation except for its validity status, confidence level etc that may be changed by its owner⁴ only.** If changes were allowed to the substantive information content of any fragment of descriptive detail it would be necessary to synchronise updates across all participating physical database instances. The communications cost and inherent complexity of such synchronisation make this something to be avoided if at all possible. This is especially true in an environment in which the WAN may need to be reconfigured at short notice in response to physical redeployment or loss of communication links.
 - (3) **Principle No 6 - The data representation of each atomic fragment of descriptive detail must provide for coexistent multiple versions in order to embrace history, projection and competing versions from different sources.** The key issue here is that on receipt of new information from a range of sources there may be no way of knowing whether the new information is more or less valid than that already held. There is also no way of knowing all the uses to which the information may be put and this may affect what constitutes the 'correct' version or versions for any particular case. The only way of preserving all options to meet the needs of future applications is to retain all versions and allow each application/user to apply the appropriate filter on retrieval.
 - (4) **Principle No 7 - Distinction between different versions of the same type of descriptive detail for the same real-world object must identify the source of the information, its effective date-time, its time of reporting and other qualifying factors.** In order that applications/users can select the appropriate version(s) for their purpose it is necessary for every atomic fragment of descriptive detail to be tagged with information about source, effective date-time etc. The factors involved here are those that enable any application to filter out versions in which it has no interest.
 - (5) **Principle No 8 - Every real-world object and every version of every atomic fragment of descriptive detail including classifications must have an identifier that is guaranteed unique across all participating systems.**⁵ This is an essential pre-requisite for the maintenance of referential integrity within the global object store as a whole.
- c. **Future Proofing The Object Model.** The two principles in this group work in conjunction with those listed above to give object representations that allow the addition of unspecified further levels of detail as required without risk of

³ For convenience and brevity the term 'atomic fragment' is used for this concept as defined here: ie "A fragment of descriptive detail for which change to any part makes the whole obsolete".

⁴ Ownership of each atomic fragment of descriptive detail is vested in the combination of user role, context and database instance responsible for its creation.

⁵ Probably the simplest way to provide this guarantee across an unlimited range of autonomous system development projects and geographically dispersed databases is for the identifier to contain both a serial number and a centrally controlled identity for the serial number generator.

overlap. (problem 2.d above). Most of the difficulties faced in accommodating new requirements are due to over optimisation in the early stages of development. The principle of maintaining the independence of each atomic fragment in a fully normalised structure means that provision can be made for every envisioned complexity in a way that costs next to nothing until it is exploited. The two principles and associated technical justifications are:

- (1) **Principle No 9 - All descriptive detail of real-world objects must be based on fully normalised abstract data types that can be given a range of business meanings by pointers to semantic data⁶ held within the same database.** There are two technical issues here. The internal structure of each unit of descriptive detail must be fully normalised so that individual atomic fragments can conform to the five principles (Nos 5 to 8 above) needed for a stable physical schema. The use of 'pointers to semantic data' as a governing feature of the design of all abstract data types is the key device that enables autonomous systems to share the same physical schema even though they are at different stages of development. Increments of semantic data can be added to a database instance without interfering in any way with application software that is not designed to handle them. It is also possible for applications to be written with generic processing where appropriate.
- (2) **Principle No 10 - The data representation of all real-world objects should be designed with the potential to allow every degree of freedom that is present in the real-world even where there is no immediate requirement.** This is the key to future proofing. It is made possible through the use of abstract data types defined in accordance with Principle No 9 above. This permits the development of an API to support objects, which exclude unwanted detail. This is done through filters that select only data described by particular semantic data that reflects the particular interests of the simpler applications. Appropriate use of such limited objects means that applications see only those details that they are currently equipped to handle. Strict adherence to this principle together with thorough business analysis should result in semantic data that can be extended rather than replaced in response to growing user expectations. This is because:
 - (a) The principle demands a set of abstract data types that place no arbitrary limits on the level of detail they can represent.
 - (b) This in turn enables the designers of semantic data to ask "What is the worst complication that we can envision users having to cope with?" instead of "What are the users top demands and what is the most efficient way to support them?"

Persistent Storage Of Object Data

9. The principles set out above all depend on there being some underlying mechanism for the management of persistent storage for the business objects concerned extending across a variable number of localities. The following principles must be obeyed by this mechanism:
 - a. **A Common Object Pool.**
 - (1) To preserve the separate identity of each fragment of descriptive detail it is necessary to have a master index of the real-world objects they describe. Every atomic fragment of descriptive detail must have reference to just one entry in this master index.
 - (2) This master index, together with the totality of atomic fragments of descriptive detail created by all participating systems, must be seen as a single common object pool that is shared on equal terms by all.
 - (3) Physical instantiation of this common object pool is spread across an unlimited number of relational database instances. All of these must be based on the same generic schema. This schema must provide a range of abstract data types that are devoid of real-world meaning. Each atomic fragment of descriptive detail must use one of these abstract data types and be given meaning by reference to semantic data held in the same database as illustrated in paragraphs 20.b(1)(b) and 20.b(2) below.

⁶ Semantic data, as used here, is data that comprises definitions that may be cited in order to give business meaning to other data. In most traditional relational databases such 'business meanings' are associated with table and column definitions held in a separate data dictionary. The 'other data' represents real business information and may be divided into; "dynamic data" that changes in response to business events, and "encyclopaedic data" that is subject to some formal approvals mechanism and comes from an authoritative source that may be either inside or outside the business concerned.

- (4) Objects are assembled as and when required from atomic fragments of descriptive detail within the common object pool. The assembly process for objects of each class is dependent on and must be associated with a corresponding set of semantic data. It is performed by appropriate functions within a single API available to all participants as the sole means of access to data within the common object pool.⁷ This constraint on the design of business objects serves to ensure adherence to all the principles stated above. The API is distributed in successive tranches that extend the range and behaviour of real-world objects represented. Each of these tranches will contain a set of object classes together with matching semantic data ready for loading onto all relevant database instances.

b. **Replication.**

- (1) Each database instance contains a non-exclusive subset of the totality of atomic fragments within the common object pool. Replication is the means by which both master index and atomic fragments of descriptive detail are distributed across the on-line database instances of the participating systems.
- (2) This task of replication is simplified by principle No 5 that the substantive content of all descriptive detail remains unchanged from the moment of its creation. From this principle it follows that the meaning of any reference to any descriptive detail is fixed and immutable. In such an environment referential integrity is enforced only within the common object pool as a whole. It need not be maintained either within each database instance or within each unit of replicated data.
- (3) The above simplification coupled with the use of identifiers that are globally unique means that replication can be managed so as to balance local storage against the need to pull data as needed from other database instances currently accessible across a WAN.
- (4) Each application instance uses the API to access details from the common object pool as if they were all present on its local server. Fragments not present on that server are obtained when needed from another physical database instance. It is a matter of ongoing system management to ensure that recourse to this pull mechanism does not occur for routine functions that are also time critical.

c. **Archiving.**

- (1) Dynamic management of content is vital for any database conforming to the principles set out above. Application of database 'updates' according to the 'no updates' principle No 5 will lead to an accumulation of superseded versions for which there may be no known need. In order to allow for the possibility of some retrospective investigation or analysis and make best use of available disc space these superseded versions should be transferred to an archive on some low cost media such as CD Rom. The archive chosen for this may be either local or somewhere else on the WAN. An intelligent archiving policy will ensure that the time and cost of copying back into the on-line database is commensurate with the urgency and value of the information concerned.
- (2) It is the job of the archiving mechanism on each physical database instance to ensure that the available on-line storage capacity is used for those fragments which are most relevant to routine processes and topical subject matter.

Scalability And The Generic Schema

10. Most of the problems addressed in this paper arise from scale. Traditional methods of system development just do not work on a multi-project basis. Traditional data structures are shaped by business rules. How then do we cope in a situation where these rules are a matter of conjecture and dispute?
11. The short answer is that we have to separate structure from semantics as embodied by principle No 9. This enables the structure to remain stable while the semantics are being negotiated. It is this that eases the problems of scale. It is no longer necessary to complete a comprehensive corporate data model before data sharing can begin. The range of business information that can be shared through the above mechanism can grow from small beginnings and continue to grow in line with user expectations and available funding.

⁷ The principles governing the design of this API are to be the subject of another paper based on work now in progress.

12. To achieve this we must go back to basics. All sharing of structured information depends on the transmission of data in some binary coded serial format. Sharing is possible only to the extent that information serialised and encoded at one end of a communications link can be decoded and reassembled at the other.
13. If the data is text then the structure is essentially serial and the coding and decoding is a fairly straightforward task. The only technical agreement required concerns a standard for representation of characters, such as ASCII, and a small amount of header information.
14. Moving on to deal with navigable data we find the tabular format of relational databases. This is also fairly straightforward provided that all partners share the same set of table definitions. Unfortunately this is not the case where the tables concerned have been defined by different people at different times working to different priorities. The differences are greatest where the tables have been optimised for a particular process or user views.
15. The main problem this presents is how to ensure that the semantics are consistent at both ends. In the relational model semantics are implied by the table names. Definition of such semantics is expressed informally if at all. These tables invariably reflect a singular perception of the real-world objects they purport to represent. Such singularity can make it impossible to harmonise table definitions from different projects. The more closely these tables match current user demands, the more difficult it is to achieve this harmonisation.
16. As indicated above, the only way to overcome this problem is to separate structure from semantics. This can be done by using a data structure made up of abstract data types that are devoid of business meaning. The application of any such abstract data type to real business information is achieved by including pointers to business definitions held as data within the same database. These definitions are here called 'semantic data' and the underlying data structure is called the 'generic schema'.
17. A very small number of such abstract data types will cover most of the information used for command and control and routine business operations. Provided these abstract data types are in fully normalised form they can be designed to accommodate great complexity while also supporting simple cases in an efficient manner.

An Example – The DCADM®

18. In order to illustrate the sort of abstract data types envisaged in this paper there follows a brief outline of the Defence Command and Army Data Model® (DCADM®). This is a full-scale practical implementation of the ten principles set out above. It has been developed over the last five years in response to the Army's goal of achieving a "fully interoperable CIS by the year 2010". It is the result of sustained collaboration between the Data Management teams of the Army, the Navy's CINCFLEET and the RAF's Strike Command. It has also benefited from overlap of personnel with a parallel endeavour pursued by NATO's Allied Data Systems Interoperability Agency. It is now adopted by the Army as its data exchange model and preferred data structure for all new command and control systems. It has also been adopted as the NATO C3 Information Exchange Model between future NATO procured systems.
19. The range of abstract data types to be provided in such a generic schema is a matter of design choice. Those provided in the DCADM® are based around a master index which comprises two tables. One of these has an entry for each real-world object instance. The other has an entry for each of the type definitions that may be used for classification. No useful information is contained in the master index itself. Instead every fragment of descriptive detail is expressed using one of seven abstract data types all of which have their own tables designed in full accordance with the ten principles set out in this paper. Each fragment of descriptive detail is associated with one entry in the master index and the physical data structure itself does not impose any constraint on the range of such descriptive detail that may be associated with any single entry.
20. The choice and internal design of the seven abstract data types for descriptive detail has been influenced by the needs of transaction processing in the highly unpredictable circumstances of battle. This follows from the Army's policy of "design for war, adapt for peace". As one would expect with designs that are "devoid of business meaning" it will be seen that there is nothing particularly military about these designs. Indications so far suggest that they will prove to be relevant across the whole range of resource management in the broadest sense. The general nature of these seven abstract data types is set out below:

- a. **Classification.** Any instance in the master index may be classified in many ways. This is done through an unlimited number of 'isa'⁸ references. Each of these:
- (1) Has its own source, date-time, status etc in line with principle No 7.
 - (2) Makes reference to a type definition represented by a row in the master index thereby implying that, according to that source and at that time, the instance concerned had or will have all characteristics and behaviour that may be defined for that type. These type definitions are defined within an inheritance structure in which:
 - (a) There is an extensible range of functional viewpoints. Each of these is represented by a classifying scheme that comprises a list of type definitions that serve to invoke different patterns of inheritance.
 - (b) Each of these type definitions is represented within the master index and may itself have descriptive detail based on any of the standard abstract data types.
 - (c) Each classifying scheme may be designed such that all of its member type definitions inherit characteristics from a broader type definition within a higher level classifying scheme.
 - (d) In addition to this hierarchic inheritance there is provision for cross-classification giving multiple inheritance.
- b. **Breakdown.** This abstract data type is designed to hold decomposition/aggregation structures. Because the abstract data type itself is devoid of any business meaning it is best understood as a family of associations between master index entries in which:
- (1) There is a root which defines:
 - (a) The master index entry (real-world object) being described. This may be either type or instance.
 - (b) The overall business meaning and context for the whole family of associations including source, date-time status etc. This is by reference to a root breakdown type defined within the extensible body of semantic data.
 - (c) The source, effective date-time, status etc.
 - (2) The root is detailed with a range of child nodes that serve as associative references to other master index entries. These are grouped according to their business meaning which is indicated by reference to a substructure breakdown type. These substructure breakdown types are also defined within the extensible body of semantic data. They are constrained for use under root nodes described by a specified root breakdown type.
 - (3) Each of these associative references may:
 - (a) Include a quantity.
 - (b) Be further detailed with a range of associative references to other master index entries. These subordinate associations are also grouped according to their business meaning or purpose. As for the first level this meaning is indicated by reference to a substructure breakdown type defined within the extensible body of semantic data.
 - (c) Have its own source, effective date-time status etc. Where not shown these are inherited from its parent associative reference or root.
 - (4) The design of this abstract data type allows any of the above associative references to be defined with a breakdown into further detail to any depth. All substructure breakdown types defined for use at these lower levels are constrained for use as detail of a single specified substructure breakdown type.
- c. **Property.** This abstract data type is essential but of less importance than those dealing with classification and breakdown. It comes third in the list because its seductive simplicity comes with a misleading and dangerous

⁸ These are references to a type definition whose characteristics are deemed to apply to the instance thus classified.

similarity to the attributes/columns of the relational data model. Each such property represents some inherent characteristic of the real-world instance or type concerned. The internal structure of this abstract data type is simplicity itself. It is just a character string with no reference to any real-world object either explicit or implied. This is why it does not equate to an attribute/column in a relational structure which can be a foreign key with a host of hidden implications. Having no reference to any real-world object, the DCADM[®] property abstract data type must not be used to represent classifications or associations with other real-world objects. These are covered in a more prescriptive and powerful way by the classification and breakdown abstract data types described above. Any attempt to do so would deprive the database of a vital linkage. If a property is numeric then it can be used in calculations but otherwise all an application can do with such properties is display them on a screen for the benefit of human users. In addition to the character string each property contains:

- (a) A Master Index reference to the real-world type or instance described.
 - (b) A reference to a business meaning defined within the body of semantic data.
 - (c) The standard filter factors of source, date-time, status etc.
 - (d) An optional reference to a qualifying statement based on structured text with embedded references as described in g below.
- d. **Resource Accounting**. This abstract data type deals with the recording of what belongs to what in an administrative sense. Its main applications are for command/organisational affiliations and holdings of equipment, personnel and other resources including standard fittings, load lists etc. It uses the principles of double entry bookkeeping to track the transfer of things within an open-ended network of independently accountable units. It includes representation of the administrative life history of the resource concerned.
- e. **Location**. This abstract data type deals with descriptive detail concerning the location of real-world objects with respect to specified spatial frames of reference such as WGS84 (standard lat/long co-ordinates).
- f. **Shape**. This abstract data type deals with descriptive detail in the form of shape definitions. It covers compound shapes as well as the basic forms of line, area and volume.
- g. **Structured Text With Embedded References**. This abstract data type deals with complex statements, which contain passing reference to real-world objects and types represented in the master index. This avoids much data duplication but has two more important advantages over free format text:
- (1) The embedded references whose implications are precisely defined within the database are much more reliable than names cited in free text.
 - (2) A link is established between real-world objects and all the statements in which they are mentioned.

Interfacing With Non-Conformant Data

21. The benefits claimed for the approach to system development advocated in this paper depend on the principle of using a single generic schema for the databases of all participating systems. The DCADM is just one such schema. It is however the result of extensive study and much broadly based collaboration. As such it provides a good test for the general principles involved.
22. However inspired the chosen single generic schema may be there will be some applications in which the benefits of optimised persistent data storage appear to provide a powerful argument for 'going it alone'. Before coming to such a conclusion it is important to ensure that these benefits are real enough to outweigh the consequent collateral costs arising from:
 - a. Import and/or conversion of data that originates in a business process supported by another computer system.
 - b. Export and/or conversion of created data needed for use by business processes supported by other computer systems.
 - c. Future changes in the business process concerned.

23. The superficial attraction of 'going it alone' must be tempered with a hard-nosed assessment of these collateral costs. Where this has been done and the case for a specialised data structure is well made it becomes necessary to provide efficient implementation of interfaces between the narrowly optimised non-conformant data and the less optimised but conformant and shareable data used and created by less specialised applications.
24. One key factor in the design of such interfaces is the principle that "The fewer the interfaces the lower the cost". Growing user expectations tend to make this factor more critical with time. The general approach advocated in this paper offers to all application developers an assurance that they can discharge their contractual and implied obligations for data sharing in a straightforward way with predictable costs that are under their own control. What they need to do is specify the semantic data needed to map their own data onto the abstract data types of the chosen generic schema and provide two-way conversion routines to match. For practical reasons to do with expertise and consistency it may be best for the actual work to be done jointly by a central data authority and the project concerned. This approach has the twin merits that:
 - a. The burden of data sharing with non-standard and highly optimised designs is borne in the same place as the localised benefits of the associated performance optimisation.
 - b. Each participating system developer has only one interface to design and implement.
25. A major problem with any new order is how to cope with what are termed 'legacy systems'. In this paper the term is used to mean any system that does not conform to the principles set out in this paper. The approach advocated in this paper is clearly a 'new order' that conflicts with established practice. As such there is an onus on its advocates to determine how to provide efficient transition/interface with systems that are part of the legacy with which it has to work.
26. A similar problem arises with COTS⁹ packages. Here there is the further problem that the package vendor may be unwilling to release details of the data structure on which the package is based. From this point of view any such 'legacy system' or COTS package is just another non-conformant data structure. It makes little technical difference whether the non-conformance is due to history, the exacting demands of a particular process or the attractions of a particular COTS package. The necessary single interface is achieved by specifying the semantic data needed to map its information onto the abstract data types of the chosen generic schema and providing two-way conversion routines to match. In the case of COTS packages the only practical way to get this done may be to get the vendor to write the conversion routines under some mutually acceptable contractual arrangement.

Effect On The Task Of Application Development

27. From the viewpoint of application development the principles described in this paper mean the replacement of a 'traditional' database by shared access to a common object pool. This in turn means that methods, tools and API associated with an underlying RDBMS are no longer relevant for the purposes of application development. Instead, the design and implementation of individual applications needs to be done in terms of the abstract data types defined within the single generic schema. The design of the required new methods, tools and API is also heavily dependent on the design of these abstract data types. There are, however, some aspects of these that are influenced by the underlying principles as follows:
 - a. **Methods.** The principles set out in this paper have a significant effect on the process of application design. There are two main ways in which this impact is felt as follows:
 - (1) The traditional roles of logical data modelling and database design as embodied in structured methods like SSADM are displaced by object modelling and the definition of semantic data. The data representation for business objects of each class needs to be defined as a specialisation of one or more of the basic structural patterns defined as abstract data types. The 'specialisation' is achieved through in-built references to semantic data. It is the need for such references that drives the definition of semantic data. For each set of abstract data types, ie for each generic schema, it is necessary to develop a methodology¹⁰ that will incorporate review by domain experts and/or user representatives. It is clearly helpful if ways can be found to do this with familiar notations and tools by the application of carefully defined rules and guidelines.
 - (2) When dealing with a single project database the normal form of access is through views which mask out unwanted columns as well as selecting relevant rows. When dealing with a common object pool as described in

⁹ Commercial Off The Shelf.

¹⁰ In the case of DCADM[®] such a methodology is under development at the time of writing.

this paper the normal form of access is through filters that mask out unwanted objects and/or constituent detail. It is incumbent on the application designer to specify these filters in a way that accurately reflects the needs of the business process being supported.

- b. **Tools.** For each set of abstract data types, ie for each generic schema, facilities need to be provided for:
 - (1) End user query based on each abstract data type.
 - (2) The recording and presentation of semantic data.
 - c. **API.** The API is needed to provide access to all defined behaviour in respect of business objects stored within the common object pool. To do this it needs to have three layers as follows:
 - (1) **A Business Layer.** This is the sole published interface and it comprises the business objects and public behaviour called for by the first of the principle No 1. This layer is to be extended progressively in parallel with the progressive extension of semantic data to cover a growing range of business objects.
 - (2) **A Generic Layer.** This is entirely private. It contains behaviour to support the abstract data types defined within the generic schema. It is invoked by the Business Layer as the sole means by which the business objects make access to the common object pool. Unlike the Business Layer, this Generic Layer is tightly bounded.
 - (3) **A Primitive Layer.** This is also private. It contains all DBMS specific behaviour needed by the Generic Layer. Such behaviour is separated out in this way to facilitate support of the Generic Layer on a range of different DBMS products. This layer is highly finite. In the case of the DCADM[®] this layer is required to support just 47 tables.
 - d. **Skills And Productivity.** The basic skills of business analysis are applied in exactly the same way as for any single project. However there is a learning curve while experience is built up in the exploitation of the different abstract data types provided by the generic schema in question. Aside from this learning curve there are other effects including:
 - (1) A need at the object modelling stage for an understanding of the abstract data types defined within the generic schema coupled with sufficient imagination to see how they apply to the business information concerned. In effect the abstract data types replace the relatively simplistic concepts of row, column and foreign key that are offered by the relational paradigm.
 - (2) It is a contention of this paper that a somewhat larger set of richer concepts can be used effectively with greater consistency than the more basic ones which demand much more in the way of creative design. This ease of use should be reflected in a much more consistent match with the real-world objects than is usually achieved with the relational model together with a gain in productivity. This is because real-world objects have an awkward habit of proving more complex than a simple list of data elements and this can involve considerable rework as the design matures.
 - e. **Performance.** Another concern of application design is performance in respect of the throughput and response time on computer equipment of particular power. The principles advocated in this paper are intended to build interoperability right into the heart of the application design. This should make it easier to achieve a clean design which can be tuned for optimum performance. The use of a small set of abstract data types means that most of the hard work involved is generic in nature and is performed by the API with a very high reuse factor. For this reason it should be possible to achieve much better overall performance than would be possible with a more traditional design with piecemeal solutions to specific requirements for information exchange. Of course it cannot be expected that a flexible system meeting a host of obligations for interoperability, future proofing, etc will match the performance of a highly optimised system working on its own preferred task or a very simple system that leaves most of the work to its users. However it must be remembered that interoperability is the goal.
28. The whole process of application design is clearly influenced by the design of the generic schema. It is necessary to take each information requirement and obtain sufficient understanding to:
- a. Determine if it or something similar is already covered by existing business object classes (and matching semantic data).

- b. If not already covered then determine which of the available abstract data types is applicable.
 - c. Establish its dependency, if any, on other information requirements or existing business object classes (and matching semantic data).
29. To proceed further it is necessary to use the above understanding to identify information types comprised of groups of similar and interdependent data elements. The results of any soft systems analysis and other statements of requirements may be helpful here.
30. Having this API as the sole means of access to the common object pool means that its Business Layer must be subject to continuous extension as a service in response to the needs of application development projects as they emerge. In most cases application developers should have no need to understand or access the semantic data since they will work entirely with the equivalent class definitions of the Business Layer.
31. There may be exceptions to this general rule where it is considered appropriate for applications to include generic processing. Such decisions are part of the design of semantic data and are dependent on the nature of the abstract data types provided within the generic schema. The following two examples are from the DCADM®:
- a. Where it is deemed useful to allow authorised users to add a new type of further breakdown for local and temporary use.
 - b. Routines that enable users to see properties of all types distinguished by property name and description extracted from the referred semantic data.¹¹

Management Issues And Other General Observations

32. To overcome the problems outlined at the beginning of this paper it is not only technical issues that have to be addressed. Interoperability between systems requires a measure of co-operation between projects. To avoid the cost and difficulty of bilateral arrangements it is necessary for this co-operation to be aided by clearly focused central co-ordination. This in turn depends on a widely understood and deep-felt conviction that the sharing of information is a critical factor in the fulfilment of corporate objectives. The work on which this paper is based has benefited from strong backing from the top due to a clearly identified need and the existence of formal responsibilities for data management.
33. Much of the information to be shared will comprise what may be called 'encyclopaedic data'. This is data of a descriptive nature that is widely used for reference and is subject to formal approval. For reasons of consistency there are usually some well known central authorities for such data and it is necessary to employ formal change control procedures. The principles set out in this paper are readily applied to such data.
34. Information sharing of the sort addressed by this paper cannot happen without some central data authority. The following is suggested as a practical split of responsibilities between this central data authority or service and the individual projects:
- a. **Central Data Authority Or Service.** Responsibilities fall to the central data authority or service for three reasons as set out below:
 - (1) The need for arbitration between the different perspectives of individual projects and the corporate interest. Under this heading there are the tasks of:
 - (a) Assisting with and approving the design of business objects based on proposals and needs expressed by individual projects.
 - (b) Negotiating definitions for semantic data.
 - (2) The need for scarce expertise. Under this heading there is:
 - (a) The design of abstract data types.

¹¹ It is intended that a comprehensive set of principles for the process of application design will be provided in a subsequent paper.

(b) The design and implementation of the API including the incorporation of support for approved business objects.

(c) General assistance to projects in their tasks as listed in b below.

(3) The need for controlled distribution. Under this heading there is:

(a) Control and distribution of the master generic schema.

(b) Distribution of approved semantic data and associated API libraries.

(c) Distribution of approved encyclopaedic data.

b. **Individual Projects**. The chief responsibilities of individual projects in this respect are:

(1) The development of business object models in accordance with the principles stated in this paper.

(2) The specification of mappings between non-conformant data and the generic schema.

(3) The implementation of two-way conversion routines including those which are provided by COTS package vendors as part of a contractual obligation.

(4) The definition of the project's encyclopaedic data and data definition requirements.

35. There can be no doubt that success in any such venture will depend on close co-operation between individual projects and the central data authority or service and that this will involve much joint work at the detail level. As with any endeavour involving people from different organisations it is also essential for the whole activity to be driven within a framework of effective programme management.

36. In very large organisations like the Army, and above it the MOD and NATO the responsibilities of the 'central data authority or service' need to be shared across a formal structure of working groups and steering committees empowered with the necessary authority and supported with sufficient funding for the technical work involved.

Conclusion

37. This paper set out to describe a solution to four intractable problems that impede any attempt to share dynamic information between systems that are subject to independent and asynchronous development.

38. The solution proposed is a set of principles that serve in combination to ease these problems by avoiding conflict between projects and simplifying the technical issues involved. A range of technical arguments are used to explain the contribution made by each of the principles and the way they work together.

39. Where the above stated principles score is in easing the problems of information sharing, reduced development cost and more flexible user facilities. Once the novelty has worn off and some experience has been built up there should be a significant reduction in the amount of skilled effort needed to implement the more sophisticated user requirements.

40. We now have a situation where raw processing power and memory are far more readily obtained than design and programming skills. For this reason we should be focusing on techniques such as those explained in this paper that harness this power to deliver better systems at lower overall cost.

41. The sort of generic schema required is illustrated by reference to the DCADM[®] which has been implemented by the Army and has been adopted as their data exchange standard. It is felt that the amount of practical experience that has been gained so far is enough to show that the solution proposed is workable. What is needed now is for the technical challenges faced in implementing these principles to be taken up by people with different types of expertise and different user needs so that the ideas will be further tested.

Table 1

(To be positioned as a panel display somewhere in the middle of the text at the discretion of the editor)

	Principles
1	Information to be shared must be defined as a single, coherent set of object classes with agreed behaviour but no visible data.
2	It must be possible for any real world object to be classified from many different functional viewpoints at the same time with provision for history, projection and competing versions from different sources.
3	It must be possible for a real world object to have behaviour that depends on the functional viewpoint from which it is addressed
4	The data representation of all real world objects must be at the atomic level. By this is meant that every fragment of descriptive detail that can be subject to independent change must have a distinct and independent existence.
5	Every atomic fragment of descriptive detail must remain unchanged from the moment of its creation except for its validity status, confidence level etc that may be changed by its owner only.
6	The data representation of each atomic fragment of descriptive detail must provide for coexistent multiple versions in order to embrace history, projection and competing versions from different sources.
7	Distinction between different versions of the same type of descriptive detail for the same real-world object must identify the source of the information, its effective date-time, its time of reporting and other qualifying factors.
8	Every real world object and every version of every atomic fragment of descriptive detail including classifications must have an identifier that is guaranteed unique across all participating systems.
9	All descriptive detail of real world objects must be based on fully normalised abstract data types that can be given a range of business meanings by pointers to semantic data held within the same database.
10	The data representation of all real world objects should be designed with the potential to allow every degree of freedom that is present in the real world even where there is no immediate requirement.