# Terminologies and Terminology Servers for Information Environments

S. K. Bechhofer, C.A. Goble, A.L. Rector, W.D. Solomon, W.A. Nowlan
and the GALEN Consortium
Medical Informatics Group
University of Manchester
Oxford Road
Manchester M13 9PL
seanb@cs.man.ac.uk
Tel: +44 161 275 6239

## Abstract

*Terminologies, or constrained vocabularies, are a potentially rich means of representing the metadata required for applications that have partially structured and incomplete dynamic data to describe, and exploratory and inexact queries to express. Such applications include digital libraries and multimedia repositories and software management; specific application communities include medicine and art. We propose that terminologies are ideal for meeting today's information requirements and that a dynamic terminology service is the appropriate architectural approach. We present the requirements of a terminology server, and describe the implementation and practical use of one developed by the authors that uses a Description Logic,* GRAIL, *to represent the terms. The Terminology Server has been extensively used in applications relating to medicine and is being used in the integration of diverse information sources for Molecular Biology.*

## 1. Introduction

Terminologies, or constrained vocabularies, are important as they provide a framework within which communities can communicate and express ideas in a consistent manner, and are essential where standardisation and unambigous information sharing is important. A terminology is a collection of terms or concepts with relationships between them, particularly the *is-a* or *subsumption* relationship, which is used to build a *classified hierarchy* of concepts. Classification supports notions such as generalization and specialization – descriptions can be refined, incrementally adding more detail when necessary, and queries can use the subsumption hierarchy to ask general questions. The use of terminologies is widespread within communities which have a need or desire to share standardised information, including systems for such diverse purposes as:

- classification of medical conditions (e.g. SNOMED) [7],
- descriptions of works of art (e.g. Iconclass) [33],
- library cataloguing (e.g. Dewey Decimal).

The application of terminologies is now receiving attention from the more general information management community because they appear particularly powerful for describing semi-structured information flexibly but with coherence and rigour. Furthermore, they are extensible in that data instances described by a collection of terms can incrementally collect those terms and be continually reclassified or indexed – extremely useful if we cannot always predict how we wish to describe a data instance at the outset. Thus terminologies are a promising representation for metadata for three main areas:

- *content-description of semi-structured information*: the description of content of information instances as documents or images cannot necessarily be strictly typed at the outset to fit with a pre-typed but static database schema where instances populate a predefined and complete schema. Application examples include: digital libraries and document/image archives [31], broadcast video archives [11] and open hypermedia systems;
- *the description of highly complex domains*: domains such as medicine and art are so complex, and the variability of description of instances is so great, that a static type system is impracticable without massive loss of descriptive richness [22];

- *ontologically-supported mediation between diverse information sources*: a terminology can be used to represent a common ontology that different information sources can map to. The common description must be rich enough to be all encompassing, and the mappings will be incomplete, imprecise and changeable as sources have missing data or are altered. As sources are described they should be classified with respect to one another to support imprecise cross-mappings [2].

All the above applications have similar requirements of their metadata:

- the data to be described is *complex* and frequently unstructured or semi-structured requiring a rich and expressive metadata model;
- the data is gathered and processed *incrementally* over time, collecting metadata (e.g. images collect annotations) so that the fundamental assumption that all metadata is known at data capture time is flawed;
- information is *incomplete* and *imprecise* as we cannot always predict how we wish to describe a data instance such as a document, unlike conventional databases where the data instances are strictly typed at the outset;
- data descriptions are *dynamic* and *evolutionary*. Flexible extensible descriptions are required as the same data may be reused from many different perspectives, and dynamically classified by many different, unpredictable, and possibly contradictory interpretations of the same contents, requiring data to be multiply classified as descriptions are elaborated and data is reused;
- incomplete descriptions lead to *imprecise queries* and *incomplete results* where the answer to a question might well just be an initial reduction of the search space. For example, (1) we may wish to present an example description and answer the question "retrieve objects that are similar to this" where the similarity is a metric of how closely the descriptions are classified w.r.t. to one another; (2) upon presentation of a request for an image containing a male politician, if the answer is empty we would like to be offered images of politicians as a more general alternative by relaxing the query constraints.

Terminologies can be seen to sit somewhere between the world of unstructured free text documents where there is no domain model or content metadata other than keywords, and more rigorous database schema for specific solutions, where highly coherent models exist but are inflexible and not painlessly extensible. Strictly-typed data schema, as in OODBMS give us sharing, consistency checking, indexing and query retrieval, but are inflexible. Attaching keywords to documents is a more flexible descriptive mechanism but the keywords used by different users are usually arbitrary and do not tend to converge – hence the terms do not form a standard interchangeable domain description. Recent work in Semantic Hypermedia systems proposes the reintroduction of database conceptual models to hypermedia systems [20] otherwise links are rarely typed or represent any rigorously expressed semantic model.

We suggest that terminologies are well suited to applications where the challenge is how to describe information intelligently and flexibly, how to evolve descriptions, how to reuse information by viewing it from multiple perspectives and how to search and browse for incomplete, inconsistent information using imprecise and exploratory/interactive questioning.

In this paper we propose the use of terminologies implemented through Terminology Logics (or Description Logics as they are also known) for describing metadata, and a service-oriented architecture that encapsulates the description logic in a **Terminology Server** – a resource that delivers a range of terminological services. We have built a terminology server for a clinical environment and report on our practical experiences from that effort.

Section 2 describes Terminology Logics and the requirements for a Terminology Server and Section 3 describes the architecture of a Terminology Server and the implementation for the GALEN [27] project and other programmes. Section 4 describes applications making use of the GALEN server. Section 5 discusses related work and Section 6 presents our conclusions. The paper deals primarily with the architecture and motivation for a terminology server, and where possible avoids detailed discussion of the description logic used within the implementation. Sections 3 and 4, however, make reference to the GRAIL description logic [26] developed during GALEN, as the techniques described there make use of special features of GRAIL.

## 2. Terminology Logics

Conventional approaches to managing terms pre-date computation (medical terminologies have been in use since the 18th Century) and do not in general have a great deal of underlying structure. A common method is to use a *coding scheme* – a tree-like structure of terms, with each term having an attached alphanumeric code, uniquely identifying the term. The subsumption relationship is then based on code substrings. An example of a coding scheme is shown in Figure 1. This is a simple hierarchy which is intended to represent types of cars and the features they can have. The example will be used to illustrate points throughout the paper.

**A1** Car
  **A11** Estate Car
    **A11X** Estate Car, 1000 cc
    **A11Y** Estate Car, 1500 cc
  **A12** Sports Car
    **A12X** Sports Car, 1000 cc
      **A12XA** Sports Car, 1000 cc, alloywheels
    **A12Y** Sports Car, 1500 cc
...

**Figure 1. A simple coding scheme**

Such schemes do provide a certain amount of support and have proved useful over the years, but they suffer from problems. With many modifiers, the number of terms required quickly grows, producing a combinatorial explosion and schemes that are too large. All the information in the representation is asserted, and must be given by a modeller or coder producing the representation. Because of this, the semantics can be unclear, leading to problems of interpretation. A lack of structure in the representation hampers navigation through the scheme, and associated activities such as data entry or the recording of information using the terms from the terminology are troublesome. At the same time, schemes can be too small, leading to insufficient coverage – as all the terms must be defined pre hoc there may not be a term appropriate to a certain situation.

An approach to providing a computational solution to term building and term management is to use a *compositional* representation which has a richer structure than the simple tree-like structures supported by traditional schemes. Such representations are provided by Description Logics (DLs), also known as Terminological Logics. These have enjoyed considerable attention from the Knowledge Representation section of the AI community in the last few years and there are a number of well known prototype DLs, descendants of the KL-ONE language, including CANDIDE, BACK, CLASSIC, and LOOM; for an overview see [5].

All DLs define complex entities in terms of composite descriptions made up of a limited set of elementary concepts assembled according to explicit rules. DLs can be viewed as languages obtained by term composition using recursive *term constructors*, where some terms are *concepts* (denoting a collection of instances or individuals) or *roles* (relationships between, or attributes of, concepts or instances). DLs distinguish between *primitive* concepts and *defined* concepts.

The concepts define a subsumption lattice. Primitive concepts have no characterising attributes and memberships and are placed in the lattice by the system designer. Defined concepts are placed *automatically* by a *classifier* based on their compositional structure. Subsumption relationships are determined primarily using structural rules, but classification also employs some inference based on cardinalities of relationships. Classified concept descriptions are a form of implied, more specific subtype of their base supertype, inheriting the properties of their supertype. This automatic classification allows us to incrementally elaborate and specialise descriptions (or terms) without having to worry about maintaining a consistent hierarchy explicitly. Recent work has provided a sound formal basis for several description logics along with results concerning their complexity [4].

The power of DLs is derived from the automatic determination of subsumption between compositional descriptions. If a descriptive model is constructed using highly compositional defined concepts wherever possible, we can navigate the resulting lattice to reveal, for example, the direct parents and children of any specific description and the least common parent and greatest common child of a pair of descriptions.

To return to our example, we would need to supply definitions for elementary concepts such as Car, Engine and so on, along with relationships to represent having an engine, an engine being of a certain size and so on. The composites such as Estate Car with 1000cc engine would then be constructed using these elementary concepts and relationships. The fact that this is a kind of Car would then be inferred by the classifier (along with other subsumption relationships – for instance this concept is also a kind of Vehicle with an Engine). Figure 2 gives an example of a Description Logic model for the Cars hierarchy given in Figure 1. This segment is given in pseudo-GRAIL syntax. The basic operations used here are those for introducing new concept and attribute definitions and adding new constraints. Car is a primitive concept. A defined concept is characterised by a set of attributes or role-fillers whose presence makes an object belong to this concept. A composite expression is generally of the form (topic which rel value), where topic and value are concepts and rel is a relationship. Expressions can be nested, with value or topic being a composite. A defined concept (in the style of GRAIL) could be GermanCar which is a Car with Germany as a role-filler for the role hasCountryOfOrigin. SportsCar which ⟨hasCountryOfOrigin Germany, hasWheelType Alloy⟩ is said to have a Concept topic SportsCar and two role-filler pairs called criteria. This composite defined concept is classified as a subtype of SportsCar which hasCountryOfOrigin Germany, SportsCar which hasWheelType Alloy, SportsCar, Car which hasCountryOfOrigin Germany, Car which hasWheelType Alloy, and Car.

**“primitive concepts”**
TopCategory newSub Car.
TopCategory newSub WheelType.
TopCategory newSub Country.
Country newSub [Germany, France].
WheelType newSub [Alloy Steel].
Car newSub [EstateCar SportsCar].
**“role definition (inverse not shown)”**
Attribute newAttribute hasEngineSize.
Attribute newAttribute hasWheelType.
Attribute newattribute hasCountryOfOrigin
**“role between concepts (sanctioning not shown)”**
Car hasEngineSize Number.
Car hasWheelType WheelType.
Car hasCountryOfOrigin Country.
**“defined concepts”**
(Car which hasCountryOfOrigin Germany)
  name GermanCar.
SportsCar which
  ⟨ hasCountryOfOrigin Germany, hasWheelType Alloy ⟩.

**Figure 2. A Description Logic Model in** GRAIL

## 2.1. Benefits of a Description Logic

Description logics have two functions that make them particularly attractive as models for describing semi-structured and complex information [5].

**A type system.** *Expressivity*: Description Logics make it possible to express the semantics of information systems and are often more expressive than traditional Semantic Data Models or Object Oriented data models;

*Type checking*: type refinement is provided automatically through the subsumption ordering on descriptions and hence provably correct subsumption algorithms can be used for type checking;

*Schema verification*: by checking whether a compositional concept’s description will classify we are able to verify the schema’s consistency;

*Type equivalence*: each compositional description contains only the necessary and sufficient descriptions for the concept. Hence terms that are expressed in different ways but come down to the same description are equivalent and redundancy is reduced.

The type system forms an ontology that can be browsed, queried and can drive interfaces.

**An indexing/query system.** *Automatic multiple classification*: Description Logics form a dynamic multiaxial classification scheme to support incremental elaboration and partial information. Concepts can be incrementally specialised, with the automatic classification capabilities of a Description Logic taking care of relationships between concepts. As more information is attributed to the description, the concept migrates through the classification hierarchy;

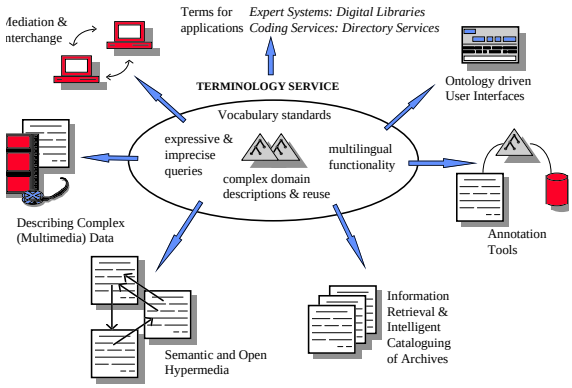


**Figure 3. The utility of a terminology service**

*Query formulation*: imprecise querying and query generalisation/specialisation is supported by a) generalising or specialising concept terms; b) relaxing or restricting cardinality constraints or c) adding or removing terms to navigate through the concept hierarchy and explore potential relationships between terms and their associated instances;

*Data exploration*: Description Logics are naturally suited for expressing queries and defining views, as the subsumption relationship can be used to organise queries automatically and views, hence supporting data exploration and query optimisation.

Description Logics have been used in a wide variety of applications including the representation of complex schemas for cars [30], software management [8] and medicine [27], data archaeology [6], mediation between heterogeneous data stores [2] and database querying [16]. However, interacting with Description Logic implementations can often prove difficult. In the past, solutions involved embedding the logic in large monolithic systems with all the attached problems of maintenance and implementation, lack of consistency, and the hampering the exchange of terms between applications. As Figure 3 suggests, a “terminology server” can be used to drive, or interact with, a number of different systems and therefore should be perceived as a separate component.

# 3. A Terminology Service

We propose a move towards a service-oriented architecture and the encapsulation of a Description Logic in a **Terminology Server** or TeS – a resource that delivers a range of terminological services. These services are used by a variety of applications such as modelling tools, query engines, data entry interfaces and so on. Figure 4 illustrates some of the services one would expect distributed servers to de-

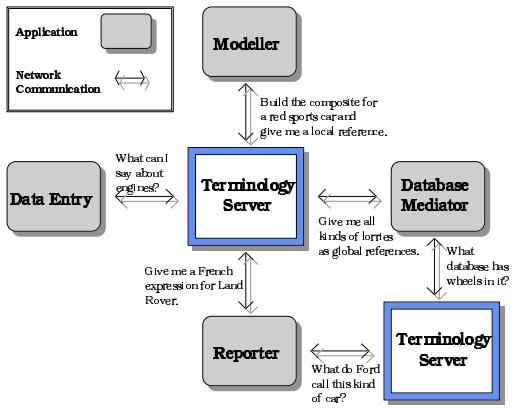


**Figure 4. Resources Delivering Terminological Services**

liver. A terminology server approach is essential if we are to build generic "plug and play" mixed systems, for example if a semantic hypermedia system is to use the TeS as a link resolution service along with conventional link following or information retrieval services [23].

### 3.1. Services

The services can be broken down into three main types: pure terminological or *concept* services, language or *linguistic* services, and *extrinsic information* services.

**Concept Services.** Along with maintaining the representation of the concepts, the services include operations which both extend and query the content of the model; these operations are commonly known as TELL and ASK [5]. Operations for extension include the introduction of new primitive concepts, the introduction of new relations, and the addition of new constraints on roles. Queries include those to determine the primitive concepts and relations; requests for the parents (generalizations) or children (specializations) of a concept definition (w.r.t. the subsumption hierarchy); usage of concepts and information about how concepts and relations can be combined to create new potential composite concepts. A description logic provides us with a *generative* framework where new conceptual definitions can be constructed by applying modifiers to base concepts. Classification operations take a conceptual definition and determine its place in the hierarchy (according to the rules defined in the logic). Such an operation may also perform validity checking that the concept is consistent and fits with any constraints the model imposes – in this way the server is providing a form of type-system (see section 2.1). Table 1 includes some examples of a concept service's operations.

| | |
|---|---|
| **ASK**<br><br>*Type Based* | *Type checking*: Is this a legal expression? Are these two definitions the same?<br>*Type exploration*: what further can be said about the expression - i.e. how can I specialize it? |
| *Index-based* | *Classification*: Where does the expression sit in the hierarchy?<br>*Query relaxation*: How can I generalize the expression?<br>*Query tightening*: How can I specialize the expression? |
| *Other* | *Language rendering*: What are the natural language expressions for this concept?<br>*Mediation & Translation*: How does this expression correspond to some external representation I have? |
| **TELL** | Introducing new elementary concepts;<br>Introducing new rules about term composition;<br>Naming concept definitions;<br>Providing maps to external representations |

**Table 1. Concept Services**

**Linguistic Services.** The server deals with terms. In order to aid interaction with the terminology, and for user interfaces, these terms should be available in representations other than the underlying one used by the description logic. Thus the server should offer linguistic services which provide the conversion to and from natural and other language expressions.

**Extrinsic Information Services.** Concept and linguistic services deal with the terminology in isolation. In addition to these, we may expect to be able to relate terms from the terminology to objects from the rest of the world. As this information associated with the terminology has no effect on the underlying classification, we refer to these as extrinsic information services. These operations are of course not completely divorced from the terminological operations – services may well use the classification in order to sparsely decorate the hierarchy and allow general querying.

We require the ability to relate terms from the terminology with terms from existing hierarchies or classifications. Such functionality is essential if a new system is to prove compatible with existing work. Domains in which a terminology server would prove useful are often those domains which already have existing terminologies – thus legacy systems must be supported [24].

In addition, terminologies can be used as mediators between information sources – the server can provide information about the concrete representation of a conceptual definition in a particular information source. These map-

pings to other representations can be seen as examples of extrinsic information services [2].

As examples, the concept (EstateCar which hasEngineSize 1000cc) could be mapped to code **A11X** from the scheme presented earlier. Alternatively we may know that details of sports cars are kept in a particular database – information about the location of this database and the format of entries can be associated with the concept SportsCar. When searching for instances of sports cars, we can use this information to determine where to look.

### 3.2. Client Applications and Concept Referencing

We can envisage multiple Terminology Servers in a distributed environment with applications coupled to a local server occasionally using other servers. A server might also support multiple concept models. This leads to two possible forms of referencing concepts:

- Persistent global references, interchangeable between servers: a collection of servers may exist, all sharing the same ontology. Concepts are exchanged between servers and must be server and session independent.
- Persistent local references for a particular server: a server may be coupled with a collection of applications using that server; such applications require fast references with reasonably tight coupling and so those references persist beyond any particular client-server session.

These assume that references are persistent and independent of any client-server session; we can also see the requirement for transient but fast session-dependent references for a particular server – for optimum efficiency a server and client application might be tightly coupled with transient references to concepts.

References provide a naming service, with local references forming a local name space (valid for a particular server) and global references a global name space (valid for all servers). These references refer to concepts in an internal representation; applications will also need to refer to them as (natural) language expressions and in terms of existing hierarchies or schemes.

By encapsulating the logic and operations as a collection of services, the terminology can be used as a resource in a distributed system; for example as a common facility in a CORBA distributed environment.

### 3.3. The Galen Terminology Server

The GALEN and GALEN-IN-USE [25] EU-funded projects have developed a Terminology Server for the support of medical applications. The Terminology Server

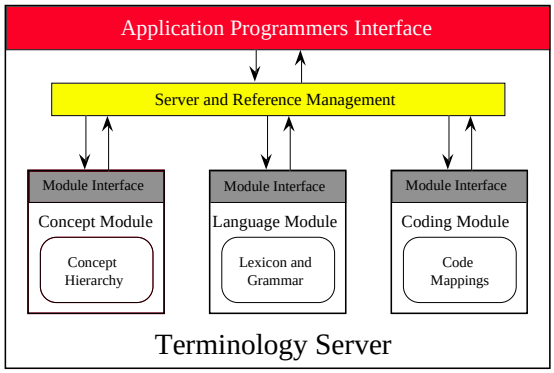


**Figure 5. The architecture of a server**

presents clinical concepts to applications which may then use the concepts to drive content-sensitive user interfaces, to mediate between medical classification systems or to act as types for medical records. The server is now being used to support applications from Biological Sciences, Art and Hypermedia.

Here we present and discuss the particular architecture of our terminology server and present a scenario to illustrate its usage. The server is implemented as a collection of modules, each with a distinct responsibility, along with a server manager. The server management layer brokers requests from the outside world, and delegates them to the appropriate modules (some requests may require services from more than one). It also deals with reference management and output scheme production as discussed in Section 3.4.

There are several modules in our current implementation, each with a rough responsibility for a class of services as described above. Figure 5 shows the three principle ones – the Concept Module, Language Module and a specific example of an Extrinsics Module for managing Clinical Coding Services.

As an example, we consider the functioning of a client tool, the purpose of which is to present a user with an interface allowing the construction of a composite concept description, and the subsequent storage of this concept in a variety of formats. An example of such a tool is the SCUI [1], which provides clinicians with the facility to append detailed clinical information to a clinical record.

We describe the dialogue which takes place between the client application and the server.

*1.* The client requires an initial entry point into the conceptual model. This may be either a collection of *global references* held by the client, or may be represented by marking or annotating concepts held by the server – thus allowing multiple clients access to the same entry points.

*2.* For each entry point X, the client is interested in how the concept can be specialised. A request “what can I say

about X?" is made to the server. This request is recognised by the server manager and delegated to the concept module. A collection of references will be returned to the client, representing the roles and concepts which can be used to construct specializations of X. These references will usually be *local references* – valid only for a particular connection to a particular server.

*3.* The client decides how to display the available options - e.g. as buttons, picking lists, or in some graphical notation. In order to display the concepts, it may need language expressions which describe them – the references are simply pointers to concepts held within the conceptual model, and do not have language explicitly attached. Thus further requests to the server may be made of the form "provide me with a language expression for concept Y". These will be delegated to the language module.

*4.* The client builds an interface based on the relationships and language provided by the server. The user selects appropriate concepts to build a particular composite. A request may then be made to the server to classify this description. Again this is passed on to the concept module, where the concept is classified – the module determines the concept's place in the hierarchy – and a reference is passed back to the client.

*5.* At this point, there are several things the client can do. We may wish to specialize further, in which case the process is repeated. It may be that the client application needs to store a persistent reference or *global ID* for the new composite. In this case, further requests to the server will be made, requiring format conversion of references. Such format conversions may require calls to modules or may be dealt with entirely by the server manager. Alternatively, the user may want to know how the built concept is represented in some other scheme. In the example of the SCUI, requests could be returned as codes from the ICD or SNOMED classifications. This requires a call to the server which is passed to the coding services module (See Section 3.6).

Code conversion is an example of inter-module communication. In order to determine a "closest matching code", the code conversion module needs to know about the parents of a concept, information provided by the concept module. All such communication is through the server manager and module interfaces.

### 3.4. Client Coupling, Reference Management and Name spaces.

The discussion above introduces the notion of concept references. The server is a *dynamic* resource. Rather than having applications hold on to conceptual definitions which contain all the required information, the server hands out references to concepts held in the Concept Module. When further information about a concept is required, the application will query the server. This does mean that the references are in some ways useless without the server, so care must be taken that information is not lost, particularly when dealing with volatile references.

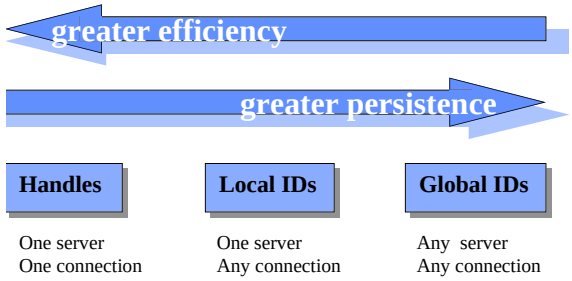


**Figure 6. Reference Types**

**Caching policies.** References and query results can be cached by both the server and clients, allowing more efficient answering of queries. This is particularly useful for operations such as language generation – in most circumstances, once the server has been queried for a string representing a concept, it is unnecessary to regenerate the string.

Caching can cause problems during the process of modelling. If new information is being added, the answers to questions, particularly those concerning the concept hierarchy, can change. Thus both clients and server must be careful to initialize and clear caches when information is updated.

As described earlier, references can be roughly split into two categories: persistent global references and local references. In the example above, the client made use of local references when building a data entry form but would require global references for persistent storage.

Our current implementation actually provides three levels as shown in Figure 6.

**Handles** are the most transient of references, and are valid for a single connection to a particular server. Handles cannot be stored persistently.

**Local IDs** are valid for a particular server and will exist over multiple connections. A local ID table is maintained by the server relating references to concepts. Local IDs can be stored, as long as the application will be connecting to the same server at a later date. Local IDs could be implemented using a table in the server relating references to concepts in the hierarchy or as references to a database if the concept module chooses to store the representation externally.

In the current implementation, there is very little difference between the efficiency of handles and local IDs. The performance of local IDs was considered good enough for most purposes, so little attempt has been made to improve handles. As the conceptual models grow, however, or if the conceptual hierarchy is stored externally, the performance of local IDs may deteriorate. We may then make use of the knowledge that handles are transient in order to employ caching or other techniques to improve the performance of handles.

**Global IDs** are interchangeable between many servers containing the same model. As each server may represent the hierarchy in different ways, Global IDs are implemented as a "decompiled" representation of the concept, which is then reclassified when presented to the server. Because of this, Global IDs are the least efficient of the references and should only be used when references are to be interchanged between servers.

**Other Output Schemes.** In addition to references, answers to questions can be requested using other representations, such as natural language or coding schemes, as illustrated in Figure 4. These formats are currently "one-way" – in our present implementations, we cannot present a natural language expression or code to the server and expect it to be treated as a concept reference. However there are operations to allow us to convert from coding scheme codes to concepts and (to a certain extent) from natural language expressions to concepts.

References, Natural Language expressions and Codes together form what are known as *Output Schemes*. A request to the server will specify the desired output scheme (or schemes) for the result. Calls will often request multiple output schemes for results. For example, a browser application will ask for children of a concept as both handles (so that further manipulations can be performed on the child concepts) and as natural language, so that a representation of the children can be presented to the user. Specifying multiple schemes in cases like this can increase performance by cutting down the overheads present in both the conceptual operations of the server and the underlying transport mechanisms of the network.

### 3.5. GRAIL sanctions.

GRAIL has been specifically devised for medical terminologies, which has influenced its range of term constructors. In contrast to some representations, certain constructs are excluded (for example, negation). GRAIL chiefly differs from its KL-ONE relatives in that it has: conceptual assertions (concept inclusions) that take part in classification; the management of part-whole relationships and limited expressions of cardinality [26]. For the purposes of this paper only one of GRAIL's differences needs explanation, as it is used by the linguistic and interface building services, namely *role sanctioning*. A sanction is an assertion that two concepts may be related via some role. Compositions of terms cannot be formed unless an appropriate sanction is present. Sanctioning serves two purposes, although they are both really different views on the same operation:

- It constrains the formation of compositional terms, ensuring that only *semantically valid* compositions are formed, preventing misnomers such as Country which hasEngineSize 1000cc.
- It provides answers to questions such as "what can I say about X?" To answer this question we examine the sanctions applying to the term X, and return an answer based on these.

Note that sanctioning has no effect on the classification process – the sanctions present or absent on a concept will not change its place in the concept hierarchy. Sanctioning is thus an operation which could be quite easily added to description logics other than GRAIL; other DLs rely on role restriction. The position of a concept in the hierarchy does effect the sanctioning process however, as sanctions are inherited. This inheritance of sanctioning allows us to place sanctions around the hierarchy in a sparse manner, allowing the construction of many compositions with a single assertion. For example, we could assert that Mammals can have a gender which is Male or Female, allowing the formation of Men, Bulls, Tom Cats etc. without having to explicitly assert that the relationship holds for Humans, Cattle, Cats and so on.

GRAIL supports two levels of sanctioning – the first represents general relationships while the second represents those things that can really be formed. To use a medical example, we may say that in general, Conditions occur in Body Parts. This is not to say that every condition can occur in every body part, but it is useful to be able to express the idea. Then at a more specific level, we can assert that Fractures occur in Bones, allowing the formation of a fracture of a bone. Sanctioning is hierarchical – a high level sanction must be present before a low-level sanction can be asserted. The levels of sanctioning provide flexible control over the formation of compositional concepts.

### 3.6. Server Modules

Here we discuss the core modules of the server in more detail.

**Concept Module.** The Concept Module, as it is responsible for the core terminological functionality of the server, includes the maintenance of the representation of the concept hierarchy and support of classification operations as

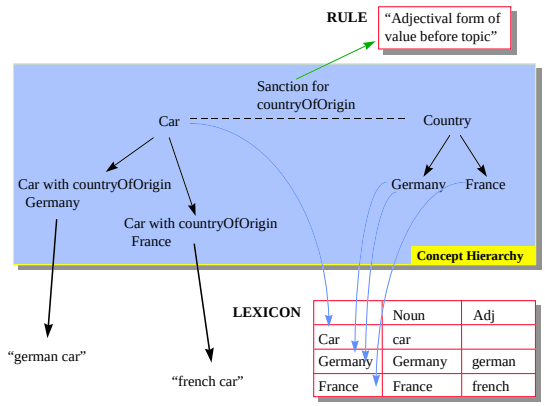


**Figure 7. Language Generation**

discussed in Section 3.1. The hierarchy is kept as a graph or semantic network, with nodes representing concept definitions and arcs represent relationships such as subsumption (the specialisation or "is-a" relationship), usage and so on. The network may actually be held in an external database, as in K-Rep [17].

**Subsumption test algorithms.** Classification depends primarily on a subsumption test – a test which determines when one concept definition is a generalization of another. The current implementation of the concept module uses a *normalize and compare* or *structural* style of subsumption algorithm [35]. Such algorithms have been shown to have problems with incompleteness – the classifier may not spot all subsumption relationships that hold. The algorithms are sound however, in that any subsumption relationships determined by the classifier will hold – and research is ongoing in the development of *tableaux-calculus* based algorithms for classification which exhibit better behaviour with respect to completeness [4].

Our architecture will enable us to replace the subsumption algorithms without any visible difference to client applications.

**Language Module.** The Language Module is responsible primarily for the generation of natural language expressions for concept definitions. Language generation uses a lexicon containing words or phrases for elementary concept definitions, along with grammar rules which describe how phrases should be combined together. The concept hierarchy is used extensively, allowing the addition of general rules which describe phrase construction for a large number of situations [34].

Figure 7 shows the workings of the Language Module. The central grey area represents the conceptual hierarchy (as stored and represented in the Concept Module). There are some elementary concepts such as Country, Car, Germany, along with some composite definitions (which will have been placed in the hierarchy by the classifier). Concepts can have attached to them specific lexical information, such as noun and adjectival forms. We have also shown a *sanction* – a constraint allowing the formation of a composite sanction by asserting that it is valid to form a composite using the concepts and relationship. The rule associated with the relationship hasCountryOfOrigin describes how to form a natural language expression from an expression using the relationship. The attachment is actually made to the sanction. Using the rule and the basic words in the lexicon, we can construct a phrase corresponding to the concept Car with countryOfOrigin Germany, i.e. "German Car". If an additional country France is introduced into the model, in order to ensure that phrases are produced for composites using the term, we need only provide the appropriate forms of phrase for the concept itself.

**Generic Extrinsic Information Module.** The Extrinsic Information Module allows us to attach information which is in some way "external" to the conceptual model. This includes information which has no terminological significance, but which should be associated with particular conceptual definitions. The model is used as a framework upon which we can hang useful facts, exploiting the inheritance hierarchy to decorate the model sparsely.

An example of the use of extrinsics is a drug interaction model produced as part of PEN & PAD [22]. A taxonomy of drugs was defined, and using extrinsic information, known interactions between different classes of drugs were recorded. This information was not considered to be important in terminological terms, but it is useful to be able to rely on the hierarchy for retrieval.

Various policies for retrieval of extrinsic information can be used including:

**Exact** Only information attached to the concept in question is returned;

**Lowest** The hierarchy is traversed in a breadth first manner from the concept, returning the first pieces of information encountered;

**All** All information attached to the concept or any of its ancestors is returned.

**Coding Services Module.**

The Coding Services Module was built for the specific purpose of relating the GALEN conceptual model to existing coding schemes for medicine [24]. It is in fact a specialised example of the more general Extrinsic Information Module. Coding Schemes are a limited form of terminology which generally consist of a collection of terms arranged in a (single-axial) hierarchy, with all relationships between terms explicitly asserted.

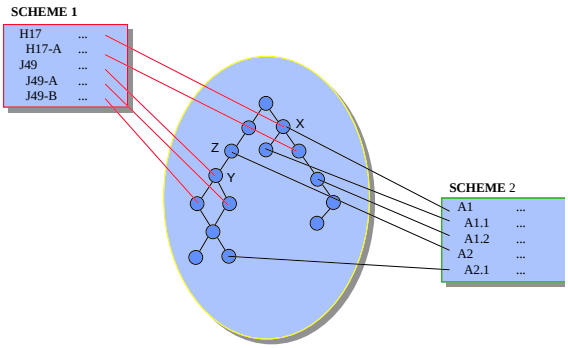


**Figure 8. Code Conversion**

If a coding scheme and the conceptual model held in the server have similar coverage, we can provide *mappings* from one to another, indicating concepts which are represented in both models. This maintenance of the mappings (along with associated functionality) is the responsibility of the Coding Services Module.

Once a collection of mappings have been defined, we can perform various retrieval operations. For example, given a code, we can ask for the concept(s) to which it has been mapped. More interesting, however, is the conversion in the opposite direction, providing codes for concepts. Given a concept X, we can ask whether the concept has an associated code mapped to it or a related concept, using one of the policies described above.

This is particularly useful when the conceptual model has more coverage or more depth than the static coding scheme. As an example, the coding scheme may not have an entry for green estate car but may have one for estate car, which is a parent of the term we need coded.

When more than one coding scheme has been mapped, an extension of this technique can provide *mediation services* between the two representations, as shown in Figure 8. The oval area represents the conceptual model held in the server with the coding schemes on either side. Starting with a code in **Scheme 1**, we can map into the representation, then out again to **Scheme 2** – even when there is no direct match. For example, starting with code H17, we can map to concept X. This has an explicit mapping to code A1 in Scheme 2, providing us with an easy conversion. Starting with code J49, and mapping to concept Y, we find no direct correspondence with **Scheme 2**. We then look up the hierarchy, and find that concept Z (a parent of Y) does have a mapping to code A2 from **Scheme 2**, and this provides us with our conversion, or "nearest corresponding code".

Traditional methods of code conversion involve crafting conversions between each pair of schemes. For each new scheme, conversions must be provided into every other scheme. With the technique described above and the use of the conceptual model as an *interlingua*, we need only provide a single mapping to and from the central representation for each coding scheme.

**Scheme Validation.** The use of mappings as described above can assist in the process of *validating* a coding scheme. If a scheme has been provided with a mapping to a representation held in the server, we can check that the subsumption relationships in the scheme are coherent, i.e. that if terms occur as parent and child in the scheme, then the corresponding terms in the model are also ancestor and descendant.

### 3.7. Application Programmer's Interface

Access to the server is via a well-defined Application Programmer's Interface, available as a library of C functions. The `xdr` notation – a data representation language suitable for remote procedure calls which forms the basis of Sun RPC – is used to describe the underlying concrete representations and to define operations for serialising and deserialising data as it is transmitted over a network. We have simply used `xdr` for data encoding, and have implemented a simple remote procedure call on top of this. The use of `xdr` ensures the availability of a C interface, and solves some of the problems caused by differences between data representations on different platforms. However, `xdr` is quite dated, and we are considering a move to a system such as CORBA.

The notion of a terminology server fits well into the CORBA framework – by encapsulating the logic and operations as a collection of services, the terminology can be used as a resource in a distributed system. CORBAMed, a subgroup of OMG, will be putting out an RFP on terminology services later this year – the aim being to specify the services that a CORBA compliant Terminology Server must support.

Client applications have been built in C and Smalltalk on both PC and Unix based platforms, and have communicated successfully with the server over both local- and wide-area networks. Work is also being undertaken on a Java binding to the C API.

## 4. Case Studies

To illustrate the use of a server and to highlight interesting areas of open questions and further work, we present two example *case studies* which provide real examples of the server technology in use. The first deals with the use of the server and its terminological logic to drive the data entry and query process. The second is concerned with the use of

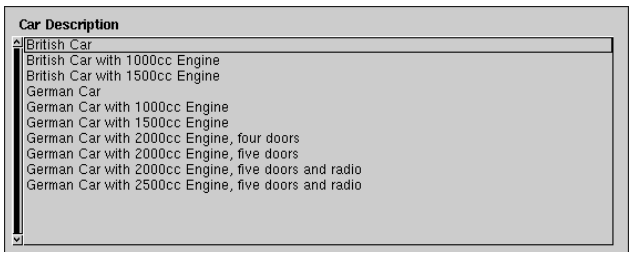


**Figure 9. A Picking List Interface**

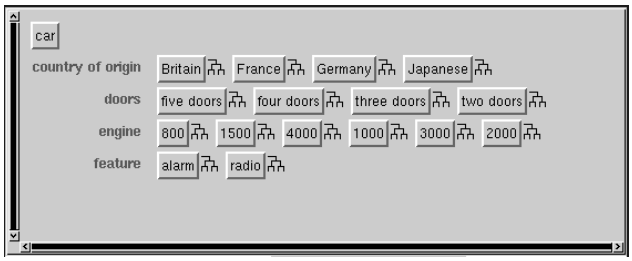


**Figure 10. A Compositional Data Entry Interface**

the server to aid in the co-ordination of the building of large scale traditional static terminologies.

### 4.1. PEN & PAD

Several applications have been built which rely on server functionality to support data entry. An example is PEN & PAD, a product which has evolved from experiments with user interface requirements for clinical data entry [22] and now forms part of the latest version of a major computer package for General Practitioners [15].

The application relies on the notion of controlled data entry, where interfaces are built according to the information present in the conceptual model. In particular this constrains the construction of complex expressions, ensuring that incoherent expressions cannot be generated.

As discussed in Section 2, GP systems had previously used static coding schemes such as SNOMED to represent clinical information. Interfaces are generally based on the notion of a *picking list*, as shown in Figure 9 where a selection of alternatives are offered to the user – note the duplication of modifiers (a by-product of the underlying structure of the representation). When several modifiers are present, many similar terms will have to be coded and presented to the user, often causing confusion.

The solution to this is a user interface that allows the *compositional* construction of terms, as shown in Figure 10.

Figure 10 shows an interface which uses controlled data entry, and allows construction of the same terms in a *compositional* manner. This second interface presents a great deal more information to the user – the picking list has only allowed a small subset of the possible combinations. If it were to contain all combinations, the list would be so long as to be unusable.

Once expressions have been built up and classified (as described in Section 3.3, the server is used to provide language expressions and codes from existing systems. These references, codes or language can then be stored in external databases for later retrieval.

The emphasis here is very much on data *entry* and capture. The added value gained from the use of the server is that language and codes can be produced quickly and found without having to resort to the usual methods of navigation through such systems which can be troublesome, as described earlier.

### 4.2. GALEN-IN-USE

GALEN-IN-USE [25] is a follow on project to GALEN [27]. The intention in GALEN-IN-USE is to use a terminological model to aid in the building of the next generation of coding schemes for medicine. Although the ultimate goal is to provide clinical terminology via a dynamic server, GALEN-IN-USE goes some way towards achieving consistent hierarchies which are to be delivered via traditional means.

GALEN-IN-USE faces several interesting challenges, many concerned with the *modelling* process. The project requires a conceptual model on a large scale – current estimates put the size of the model in the region of 10,000 concepts. This is simply the number of named and elementary concept definitions – as the representation is *generative*, the actual number of concepts which can be described using the model is huge. Contrast this with the size of most knowledge bases which seldom rise above the hundreds.

Tools are thus required which help in the various activities associated with building and maintaining the model. These currently take the form of a variety of *knowledge browsers* providing both textual and graphical representations of the structure of the model.

The project is collaborative, involving several international partners on the modelling side. Support is thus required for the process of *harmonising* sections of model. This occurs at both the *technological* level, where tools are needed to allow verification and consistent integration of new knowledge and at the *management* level, where modellers at different sites must be co-ordinated. This is supported through a variety of means including e-mail and WWW forums.

Population of the model is helped by the use of existing corpora (which will ultimately be mapped to concepts in the central model) as sources for terms and constraints. Some

of this process has been automated, with natural language processing techniques being used on the rubrics associated with codes. In addition, much use is made of metadata and *templates*. Intermediate representations are used to insulate modellers from the complexities of the underlying representations. Rubrics are "dissected" into an intermediate format which can then be translated into GRAIL expressions using tools such as the SPET and TIGGER [29].

The modelling process is often *interactive*, requiring a certain level of performance from the classification engines which form the heart of the terminology server. Early implementations were built in Smalltalk – a language supporting rapid prototyping and the easy construction of user interface tools. However, as the models grow larger, the performance of the Smalltalk implementations becomes unacceptable. Alternative implementations of the core conceptual functionality have been built in C++ [28]. The modular nature of the server internals allowed the replacement of the classification engine with a minimum of disruption.

## 5. Related Work

A considerable amount of work has been undertaken in the area of Description Logics that has largely fallen into two divisions: theoretical work with little or no implementation and practical systems with little view to extensible services or the application to large scale concept models. None have adopted the approach of a terminological server; instead they concentrate on the details of the classifier rather than taking an overall systems architecture view.

Workers in the area of information integration have taken a more architectural viewpoint, for example those working within the I3 (Intelligent Information Integration) initiative. The I3 reference architecture [3] proposes four families of services; co-ordination and management, semantic integration and transformation, functional extensions and wrapping. A terminology service would appear to be orthogonal to these families and contribute to all of them: ontologies for service brokering; terms for describing the semantic content of different services; terminological inferencing etc.

SIMS [2], and Observer [19], part of the InfoQuilt programme [14], use the Description Logics LOOM and Classic respectively to mediate between heterogeneous information sources. SIMS still seems to perceive the terminology as an embedded component rather than an information service. However, Observer describes an *ontology server* – an application providing access to ontologies represented using a description logic, and mappings from terms in the ontologies to data repositories. This can be seen as a kind of terminology server as described here.

The InfoQuilt project [14] advocates the use of an ontology to describe multimedia and although does not suggest Description Logics or an ontology server as an approach one presumes from their work in integration that this is an approach that they are considering. Some forays have been made into the application of DLs to document description, e.g. CANDIDE [21], but this work is not described in detail and there is little evidence of its practical application. Likewise [18] theoretically applies a terminology logic to information retrieval but without any view on a terminological service architecture.

Farquhar et al [10] present the case for an *ontology server* that supports the ability of users to publish, browse, create and edit ontologies through the World Wide Web. The main purpose behind this is the support of collaborative development of ontologies and the process of achieving consensus on common ontologies by distributed groups. In our GALEN-IN-USE project similar support is achieved through web-based discourse tools. Farquhar et al's emphasis is on a shared ontolingua that supports assembly and reuse of existing ontologies from a repository by way of batch loading and translations from the ontology server to stand-alone applications – this is in direct contrast to our approach of a dynamic integrated terminology server serving individual concepts to client applications. From an application viewpoint, that of digital libraries, [9] discusses the need for an *authority tool* which is an application providing a knowledge service and the functionality of a thesaurus; this can be interpreted as a terminology server.

## 6. Conclusions and Further Work

In this paper we have presented the case for terminologies to be used as a means of richly representing the metadata for applications that have partially structured and incomplete dynamic data to describe, and exploratory and inexact queries to express. Applications with these characteristics are the applications of interest in this decade. Such terminologies represent ontologies that should not be embedded in client applications but should be shared and reused as a distributed resource. To make this possible the terminologies should be implemented as a service through a terminology server offering a number of sub-services: conceptual, linguistic and extrinsic. We have described a particular implementation of a service using a particular technique known as Description Logics; however, there is no reason to presume that this is the only representation for terms. Indeed, ConceptBase [13] could be considered as a terminological service where the concepts are represented in a deductive frame-based model, although it happens that DLs are particularly suited to interface building. We have shown how a terminology service can support a wide range of applications, and how one might form a useful component in a distributed information system. However, several open questions still remain.

**Tools.** Further support and tools are required for the modelling process, addressing both the managerial and technical problems associated with distributed and collaborative model building. The use of a terminological model provides added value, but there is a price to pay in that a conceptual model of the domain must be produced.

**Interacting Servers and Models.** The provision of Global IDs suggests that architectures with multiple terminology servers are possible. So far, however, our experiences have been with an architecture containing a *single* server. Global IDs are still useful though, as that single server may change. In addition, all the reference types described above are presumed to be references to concepts in a single conceptual model. How would servers interact if they had differing conceptual models with a degree of overlap? Preliminary investigations into image catalogues suggest that multiple, disjoint terminologies are required. How are these to be provided?

**Query.** So far no real attempt has been made to consider the process of retrieval. The use of the conceptual model can provide powerful query facilities, with queries being formed at abstract levels and the conceptual model providing a space around which we can navigate. For instance, if a query proves too general, we can consider specializing the concept (or possibly sub-concepts or sub-queries). Alternatively, if the query is too specific, sections of a query could be replaced with parent concepts, broadening the search space.

The use of the model for querying rather than data entry poses new problems in terms of user interfaces which are being addressed as part of the TAMBIS project [32]. In general, the level at which queries can be asked will differ to the level at which information must be specified for data entry. For example, it is generally the case that a medical Condition will occur in a Body Part, and so asking for all the instances or occurrences of Condition located in Body Part is reasonable. However, we are unlikely to allow the recording of such a concept in a record. Thus data entry and data retrieval place *different* requirements on the controlling user interfaces.

**Performance.** Expressive DLs are known to have problems with tractability. If a terminology server is to be used in an *interactive* fashion as we suggest, the performance of the classifier must be adequate. We have investigated the benefits of parallelism [28], and research is ongoing into optimisation techniques for classification [12].

**Instances.** Using a terminology as a query language requires the full integration of an instance space with the classification operations offered by the concept model. Although acknowledged widely throughout the Description Logic community, this has been an area of some neglect, with few implementations of Description Logics providing complete reasoning due to the practical problems of tractability. We are currently pursuing research in this area, and there are questions as to how much completeness is really required in the reasoning over instances in order to support practical and useful applications.

## 7. Acknowledgements

The authors would like to acknowledge the other members of the Medical Informatics Group and GALEN consortium. We would like to thank Graham Riley for his invaluable comments on this paper. This research is supported by the European Union under the Advanced Informatics in Medicine (AIM) project 2012 (GALEN) and HealthCare Telematics project HC 1018 (GALEN-IN- USE), and by the UK through EPSRC grant GR/J98820 (PAEPR), BBSRC grant BIF/05344 (TAMBIS), and EPSRC/DTI grant TCS VAMP GRH 48811.

## References

[1] L. L. Alpay, W. A. Nowlan, W. D. Solomon, C. Lovis, R. H. Baud, T. Rush, and J.-R. Scherrer. Model-Based Application: The GALEN Structured Clinical User Interface. In *AIME 95*, 1995.

[2] Y. Arens, C. Y. CHee, C.-N. Hsu, and C. A. Knoblock. Retrieving and Integrating Data From Multiple Information Sources. *International Journal on Intelligent And Cooperative Information Systems*, 2(2):127–158, 1993.

[3] Y. Arens, R. Hull, and R. King. REFERENCE ARCHITECTURE for the Intelligent Integration of Information, Version 2.0, 1995.

[4] F. Baader and B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Processing declarative knowledge: International workshop PDK'91*, number 567 in Lecture Notes in Computer Science, pages 67–86. Springer-Verlag, 1991.

[5] A. Borgida. Description Logics in Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–782, 1995.

[6] R. Brachman, P. Selfridge, L. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. McGuinness, and L. Renick. Integrated Support for Data Archaeology. *International Journal of Applied and Cooperative Information Systems*, 2(2):159–185, 1993.

[7] R. Cote, D. Rothwell, J.L.Palotay, R. Becket, and L.Brochu. *The Systemised Nomenclature of Medicine: SNOMED International.* College of American Pathologists, Northfield, IL, 1993.

[8] P. Devanbu, R. Brachman, P. Selfridge, and B. Ballard. LaSSIE: A Knowledge-based software information system. *Communications of the ACM*, 34(5), 1991.

[9] M. Doerr. Authority Services in Global Information Spaces. Technical report, Institute of Computer Science, Foundation for Research and Technology - Hellas, Heraklion, Crete, Greece., 1996.

[10] A. Farquhar, R. Fikes, W. Pratt, and J. Rice. Collaborative Ontology Construction for Information Integration. Technical report, Knowledge Systems Laboratory, Dept of Computer Science, Stanford University, 1995.

[11] C. Goble, C. Haul, and S. Bechhofer. Describing and Classifying Multimedia using the Description Logic GRAIL. In *Conference on Storage and Retrieval of Still Images and Video IV*, San Jose, 1996. SPIE Vol 2670.

[12] I. Horrocks. Optimisation Techniques for Expressive Description Logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science, 1997.

[13] Jorke, Matthias and Gallersdörfer, Rainer and Jeusfeld, Manfred A. and Staudt, Martin and Eherer, Stefan. ConceptBase - A Deductive Object Base for Meta Data Management. *Journal of Intelligent Information Systems*, 3:167–192, 1994.

[14] V. Kashyap and A. Sheth. Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies. In G. Papazoglou, M. & Schlageter, editor, *Cooperative Information Systems: Current Trends and Directions*. (to appear).

[15] J. Kirby. PEN & PAD: The Next Generation. In *Primary Health Care Specialist Group*, Cambridge, UK, 1995.

[16] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *AAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments.*, Stanford, California, 1995. AAAI.

[17] E. Mays, R. Dionne, and R. Weida. K-Rep System Overview. *SIGART Bulletin*, 2(3):93–97, 1991.

[18] C. Meghini, F. Sebastiani, U. Straccia, and C. Thanos. A model of information retrieval based on a terminology logic. In *Proc ACM SIGIR93. Pittsburg, USA*, pages 298–307, 1993.

[19] E. Mena, V. Kashyap, A. Illaramendi, and A. Sheth. Managing Multiple Information Sources through Ontologies: Relationship between Vocabulary Heterogeneity and Loss of Information. In *Knowledge Representation meets Databases KRDB '96, ECAI '96*, Budapest, Hungary, 1996.

[20] J. Nanard and M. Nanard. Should Anchors Be Typed Too? An Experiment with MacWeb. In *Hypertext '93*, pages 51–62, 1993.

[21] S. Navathe, A. Savasere, T. Anwar, H. Beck, and S. Gala. Object Modelling Using Classification in CANDIDE and its Applications. In A. Dogac, T. Ozsu, A. Briliris, and T. Sellis, editors, *Advances in Object-Oriented Database Systems*, volume 130 of *ASI Series*, pages 435–476. NATO, 1994.

[22] W. A. Nowlan, A. L. Rector, S. Kay, B. Horan, and A. Wilson. A Patient Care Workstation Based on User Centred Design and a Formal Theory of Medical Terminology: PEN & PAD and the SMK Formalism. In *Fifteenth Annual Symposium on Computer Applications in Medical Care. Proceedings of SCAMC91*, pages 855–857. McGraw-Hill Inc., 1991.

[23] J. O'Neill. *An Investigation into the use of GRAIL as a Hypermedia Authoring Tool.* University of Manchester, Department of Computer Science, 1996.

[24] P. Pole and A. Rector. Mapping the GALEN CORE Model to SNOMED III:Initial Experiments. In *1996 AMIA Annual Fall Symposium (Formerly SCAMC)*, 1996.

[25] A. Rector, P. Zanstra, and The GALEN Consortium. The GALEN-IN-USE Project. *To appear in: European Health Telematics Observatory Journal 1996*, 1996.

[26] A. L. Rector, S. K. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL Concept Modelling Language for Medical Terminology. *Artificial Intelligence in Medicine*, (9):139–171, 1997.

[27] A. L. Rector, P. Zanstra, W. D. Solomon, and The GALEN Consortium. GALEN: Terminology Services for Clinical Information Systems. In M. Laires, M. Ladeira, and J. Christensen, editors, *Health in the new Communications Age*, volume 24 of *Health Technology and Informatics*. IOS Press, 1995.

[28] G. D. Riley, J. M. Bull, and A. P. Nisbet. Parallelisation of a Semantic Network Classifier. In *2nd European School of Computer Science, Parallel Programming Environments for High Performance Computing,*, l'Alpe d'Huez, 1996.

[29] J. E. Rogers, S. W. D., A. L. Rector, P. Pole, P. Zanstra, and E. van der Haring. Rubrics to Dissections to GRAIL to Classifications. In *To appear in: MIE 97*, 1997.

[30] N. Rychtyckyj. DLMS: An Evaluation of KL-ONE in the Automobile Industry. In *KR' 96*, pages 588–596, 1996.

[31] J. Schmidt, G. Schroder, C. Niederee, and F. Matthes. Linguistic and Architectural Requirements for Personalized Digital Libraries. *International Journal of Digital Libraries (JODL)*, 1(1), 1996.

[32] TAMBIS. Tambis www home page. http://www.cs.man.ac.uk/mig/tambis/index.html, 1996.

[33] H. v. d. Waal. *ICONCLASS: An Iconographic Classification System.* Koninklijke Nederlandse Akademie van Wetenschappen, 1973-1985.

[34] J. C. Wagner, R. H. Baud, and J.-R. Scherrer. Generating Noun Phrases from a Medical Knowledge Representation. In *MIE 94*, pages 218–223, Lisbon, Portugal, 1994.

[35] W. Woods. Understanding Subsumption and Taxonomy: A Framework for Progress. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, pages 45–94. Morgan Kaufmann, San Mateo, CA, 1991.