COURSE CODE: DAM 344

COURSE TITLE: SEMANTIC DATA MODELLING.

DAM 344
SEMANTIC DATA MODELLING.

Course Developer/Writer Dr. AWODELE Oludele

Programme Leader Prof. Kehinde Obidairo

Course Coordinator Greg. Onwodi

The study units in this course are as follow:

Module 1 Concepts of Data Modelling

Unit 1 Overview of Data Modelling

Unit 2 Data Modelling concepts

Unit 3 Data models

Module 2 Semantic Data Modelling

Unit 1 Overview of Semantic Data Modelling

Unit2 Semantic Data Models

Unit 3 Semantic Data Modelling Concepts

Module 3 Areas of Application of Semantic Data Modelling

Module 1 Concepts of Data Modelling

Unit 1 Overview of Data Modelling

1.0 Introduction

Data modelling is a critical skill for IT professionals including someone who is familiar with relational databases but who has no experience in data modelling, such people as database administrators (DBAs), data modellers, business analysts and software developers. It is an essential ingredient of nearly all IT projects. Without a data model there is no blueprint for the design of the database.

Then, there are two important things to keep in mind when learning about and doing data modelling:

Data modelling is first and foremost a tool for communication. There is no single "right" model. Instead, a valuable model highlights tricky issues, allows users, designers, and implementers to discuss the issues using the same vocabulary, and leads to better design decisions.

The modelling process is inherently iterative: you create a model, check its assumptions with users, make the necessary changes, and repeat the cycle until you are sure you understand the critical issues.

2.0 Objectives

At the end of this unit, you should be able to:

- ❖ Describe what data modelling is and why it is required
- ❖ Mention types and uses of data modelling
- ❖ Describe the process of data modelling, with the aid of diagram
- ❖ Describe the outstanding data modelling methodologies
- ❖ List and explain the properties of data

3.0 Definition of Data Modelling

Data modelling is the process of creating and extending data models which are visual representations of data and its organization. The **ERD Diagram** (Entity Relationship Diagram) is the most popular type of data model. In software engineering, it is the process of creating a data model by applying formal data model descriptions using data modelling techniques.

It is a method used to define and analyze data requirements needed to support the business processes of an organization. Data modelling defines not just data elements, but their structures and relationships between them. Data modelling is also a technique for detailing business requirements for a database, and it is sometimes called database modelling because, a data model is eventually implemented in a database.

3.1 Types of Data Modelling

Data modelling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in repository so that they can be retrieved, expanded, and edited over time. Whitten (2004) determined two types of data modelling:

- **Strategic data modelling:** This is part of the creation of an information systems strategy, which defines an overall vision and

architecture for which, information systems is defined. Information engineering is a methodology that embraces this approach.

- **Data modelling during systems analysis:** In systems analysis logical data models are created as part of the development of new databases.

3.2 Use of Data Modelling

Data modelling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modelling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modelling:

- To manage data as a resource;
- For the integration of information systems;
- For designing databases/data warehouses (a.k.a data repositories)

3.3 Data Modelling Process

The actual database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity. The term database design can be used to describe many different parts of the design of an overall database system.

Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an Object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the Database Management System or DBMS.

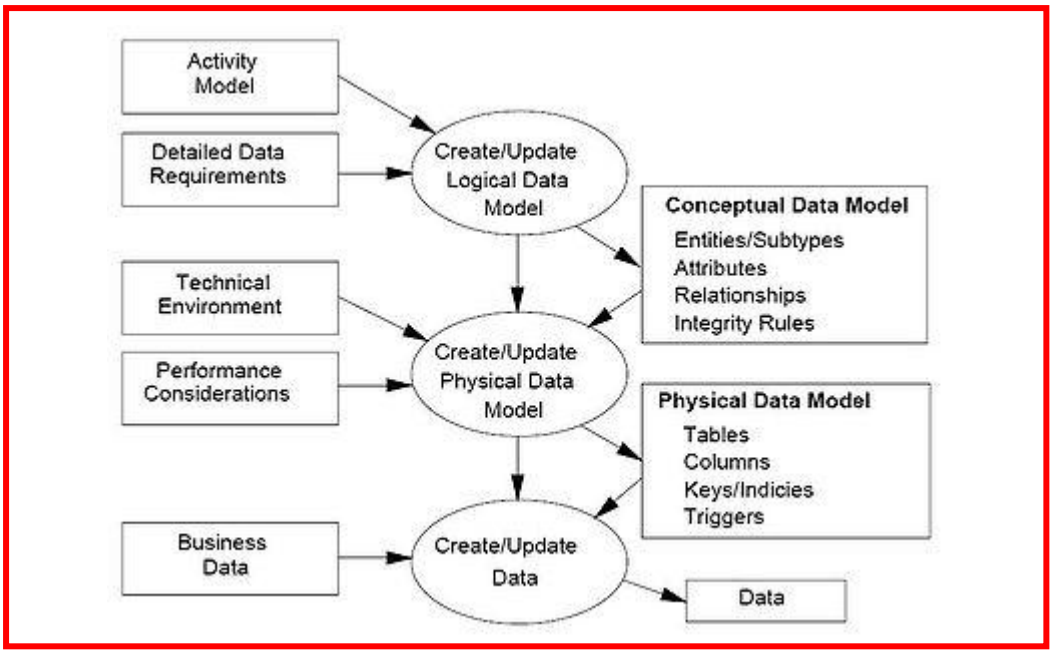

Figure 1. **The data modelling process**

The figure illustrates the way data models are developed and used today. A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.

3.4 Modelling Methodologies

Though data models represent information areas of interest, and there are many ways to create data models, according to Len Silverston (1997), only two modelling methodologies stand out:

- **Bottom-up models:** These are often the result of a reengineering effort. They usually start with existing data structures forms, fields on application screens, or reports. These models are usually physical, application-specific, and incomplete from anenterprise perspective. They may not promote data sharing, especially if they are built without reference to other parts of the organization.
- **Top-down logical data models**: These on the other hand, are created in an abstract way by getting information from people who know the subject area. A system may not implement all the entities in a logical model, but the model serves as a reference point or template.

Sometimes models are created in a mixture of the two methods; by considering the data needs and structure of an application and by consistently referencing a subject-area model. Unfortunately, in many environments the distinction between a logical data model and a physical data model is blurred. In addition, some CASE tools don't make a distinction between logical and physical data models.

3.5 Benefits of Data Modelling

Abstraction: The act of abstraction expresses a concept in its minimum, most universal set of properties. A well abstracted data model will be

economical and flexible to maintain and enhance accuracy, since it will utilize few symbols to represent a large body of design. If we can make a general design statement which is true for a broad class of situations, then we do not need to recode that point for each instance. We save repetitive labour; minimize multiple opportunities for human error; and enable broad scale, uniform change of behaviour by making central changes to the abstract definition.

In data modelling, strong methodologies and tools provide several powerful techniques which support abstraction. For example, a symbolic relationship between entities need not specify details of foreign keys since they are merely a function of their relationship. Entity sub-types enable the model to reflect real world hierarchies with minimum notation. Automatic resolution of many-to-many relationships into the appropriate tables allows the modeller to focus on business meaning and solutions rather than technical implementation.

Transparency: Transparency is the property of being intuitively clear and understandable from any point of view. A good data model enables its designer to perceive truthfulness of design by presenting an understandable picture of inherently complex ideas. The data model can reveal inaccurate grouping of information (normalization of data items), incorrect relationships between objects (entities), and contrived attempts to force data into preconceived processing arrangements.

It is not sufficient for a data model to exist merely as a single global diagram with all content smashed into little boxes. To provide transparency a data model needs to enable examination in several dimensions and views:

diagrams by functional area and by related data structures; lists of data structures by type and groupings; context-bound explosions of details within abstract symbols; data based queries into the data describing the model.

Effectiveness: An effective data model does the right job - the one for which it was commissioned - and does the job right - accurately, reliably, and economically. It is tuned to enable acceptable performance at an affordable operating cost.

To generate an effective data model the tools and techniques must not only capture a sound conceptual design but also translate into a workable physical database schema. At that level a number of implementation issues (e.g., reducing insert and update times; minimizing joins on retrieval without limiting access; simplifying access with views; enforcing referential integrity) which are implicit or ignored at the conceptual level must be addressed.

An effective data model is durable; that is it ensures that a system built on its foundation will meet unanticipated processing requirements for years to come. A durable data model is sufficiently complete that the system does not need constant reconstruction to accommodate new business requirements and processes.

3.6 Properties of Data

Some important properties of data for which requirements need to be met are:

- definition-related properties
  - *relevance*: the usefulness of the data in the context of your business.
  - *clarity*: the availability of a clear and shared definition for the data.
  - *consistency*: the compatibility of the same type of data from different sources.

- content-related properties
  - *timeliness*: the availability of data at the time required and how up to date that data is.
  - *accuracy*: how close to the truth the data is.
- properties related to both definition and content
  - *completeness*: how much of the required data is available.
  - *accessibility*: where, how, and to whom the data is available or not available (e.g. security).
  - *cost*: the cost incurred in obtaining the data, and making it available for use.

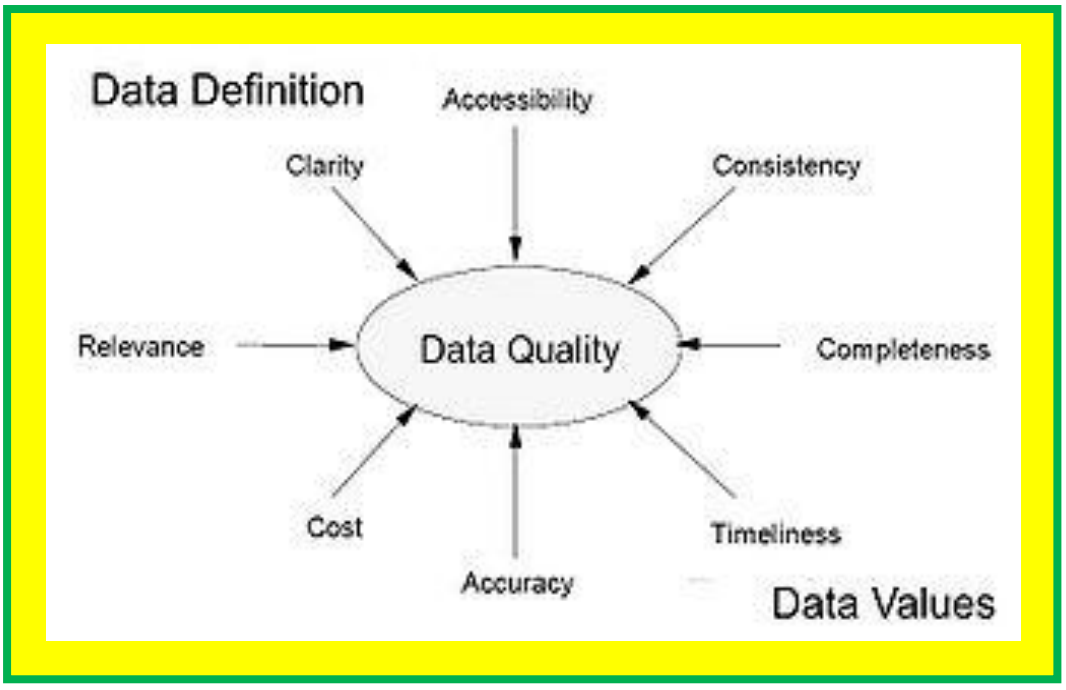

Figure 2. **Some important properties of data**

4.0 Conclusion

With the overview of data modelling, individuals and organizations can uncover hidden processes, methodologies, as well as the benefits of modelling their data, which they can use to predict the behaviour of customers, products and processes.

5.0 Summary

In this unit we have learnt that:

- ❖ Data modelling is a critical skill for IT professionals, and that, it is first and foremost a tool for communication and inherently iterative.
- ❖ Data modelling is the process of creating and extending data models which are visual representations of data and its organization.
- ❖ Some of the uses of data modelling include data management, integration of information systems, and designing of databases.
- ❖ Benefits of data modelling are abstraction, transparency and effectiveness.

6.0 Tutor Marked Assignment

1. (a) What do you understand by the term data modelling?
   (b) Explain the modelling processes and mention its methodologies.
2. (a) Simply explain the benefits of data modelling
   (c) Mention the various properties of data

7.0 Further Reading and Other Resources

E.F. Codd (1970). "A relational model of data for large shared data banks". In: *Communications of the ACM archive*. Vol 13. Issue 6(June 1970). pp.377-387.

Importance-of-Logical-Data-Model. http://www.blueink.biz/RapidApplicationDevelopment.aspx

Data Integration Glossary, U.S. Department of Transportation, August 20

Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2004). *Systems Analysis and Design Methods*. 6th edition. ISBN 025619906X

American National Standards Institute. 1975. *ANSI/X3/SPARC Study Group on Data Base Management Systems; Interim Report*. FDT (Bulletin of ACM SIGMOD) 7:2.

Semantic data modeling" In: *Metaclasses and Their Application*. Book Series Lecture Notes in Computer Science. Publisher Springer Berlin / Heidelberg. Volume Volume 943/1995.

FIPS Publication 184 released of IDEF1X by the Computer Systems Laboratory of the National Institute of Standards and Technology (NIST). 21 December 1993

Len Silverston, W.H.Inmon, Kent Graziano (2007). *The Data Model Resource Book*. Wiley, 1997. ISBN 0-471-15364-8. Reviewed by Van Scott on tdan.com. Accessed 1 Nov 2008.

Paul R. Smith & Richard Sarfaty (1993). Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools. Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.

Module 1 Concepts of Data Modelling

Unit 2 Data Modelling Concepts

1.0 Introduction

The motive of data modelling concepts is that, all developers should have skills that can be applied on project, with the philosophy that, every IT professional should have a basic understanding of data modelling. This is a brief introduction to these skills. So, it is critical for application developers to understand the concepts and appreciate the fundamentals of data modelling

2.0 Objective

At the end of this unit, you should be able to:

- ❖ Describe generic data model
- ❖ Differentiate between identifier, modifier and descriptor
- ❖ Explain different concepts of data modelling
- ❖ Explain with the aid of diagram, the syntax of common data modelling notations

3.0 Generic Data Modelling

Generic data models are generalizations of conventional data models. They define standardized general relation types, together with the kinds of things that may be related by such a relation type. The definition of generic data model is similar to the definition of a natural language. For example, a generic data model may define relation types such as a 'classification relation', being a binary relation between an individual thing and a kind of thing (a class) and a 'part-whole relation', being a binary relation between two things, one with the role of part, the other with the role of whole,

regardless the kind of things that are related.

Given an extensible list of classes, this allows the classification of any individual thing and to specify part-whole relations for any individual object. By standardization of an extensible list of relation types, a generic data model enables the expression of an unlimited number of kinds of facts and will approach the capabilities of natural languages. Conventional data models, on the other hand, have a fixed and limited domain scope, because the instantiation (usage) of such a model only allows expressions of kinds of facts that are predefined in the model.

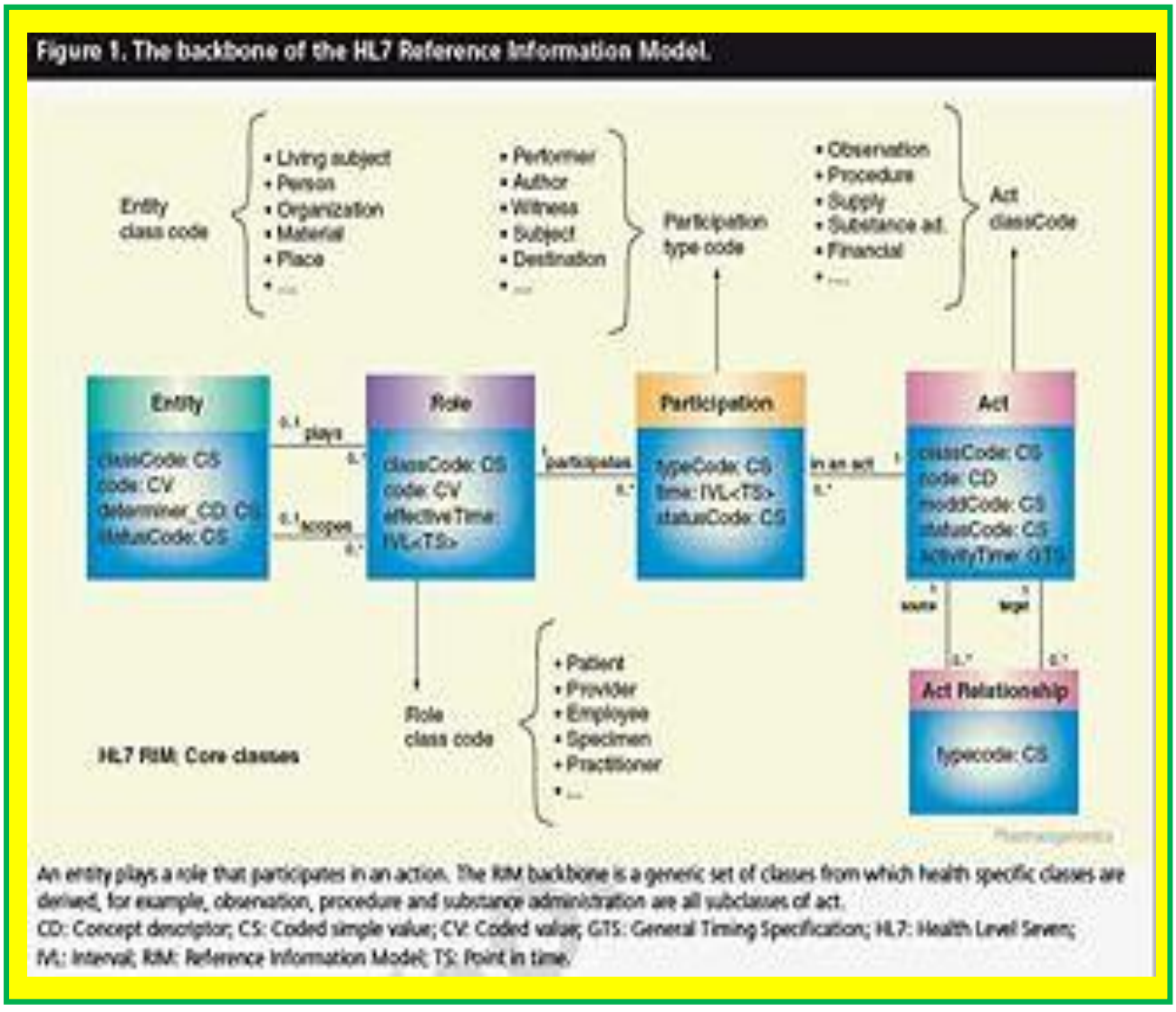

Figure 3. **Example of a Generic data model**

3.1 Concept of Identifier, Modifier, and Descriptor

Identifier (I):

Identifiers serve as the primary identification terms, or keys, necessary to uniquely identify things and events, or classes of them. They symbolically represent things and events (entities) and provide for the necessary identification of conceptual objects. Importantly, identifiers provide the skeletal structure upon which all other types of data depend. They also provide a means for explicitly defining the relationships between things and events (entities), which enables data sharing among users. Typical identifiers include: patient, account, part and purchase order numbers.

Modifier (M):

Modifiers serve as sub-entity identifiers and expand upon or refine primary identification (identifiers). As variant forms of identification, they cannot stand alone. Modifiers must be used in conjunction with identifiers to form fully qualified identification terms. They primarily are used to identify such things as: time, occurrence, use, type, sequence, etc. Modifiers have a unique data element value for each variation of the identifier addressed and can exist with one-to-one (1:1) or one-to-many (1:m) cardinality. Typical modifiers include: dates, type codes, serial numbers and revisions.

Descriptor (D):

Descriptors are non-identification data elements used to characterize entities and relationships of them. There are no logical dependencies between

descriptors and they can only exist when associated with an identifier or identifier-modifier combination. Descriptors comprise the majority of all data elements and frequently are textual or codified data element values -- data that must be further interpreted by the user to have meaning. Some descriptors are numbers capable of being mathematically manipulated, while others are numerals that do not follow strict mathematical rules. Typical descriptors include: dates, names, descriptions, codes, numeric values *and* images.

3.2 Relational Model and Relationships:

The relational model used the basic concept of a relation or table. The columns or fields in the table identify the attributes such as name, age, and so. A **tuple** or **row** contains all the data of a single instance of the table such as a person named Doug.

In the relational model, every tuple must have a unique identification or key based on the data. In this figure, a social security account number (SSAN) is the key that uniquely identifies each tuple in the relation. Often, keys are used to join data from two or more relations based on matching identification. The relational model also includes concepts such as foreign keys, which are primary keys in one relation that are kept in another relation to allow for the joining of data.

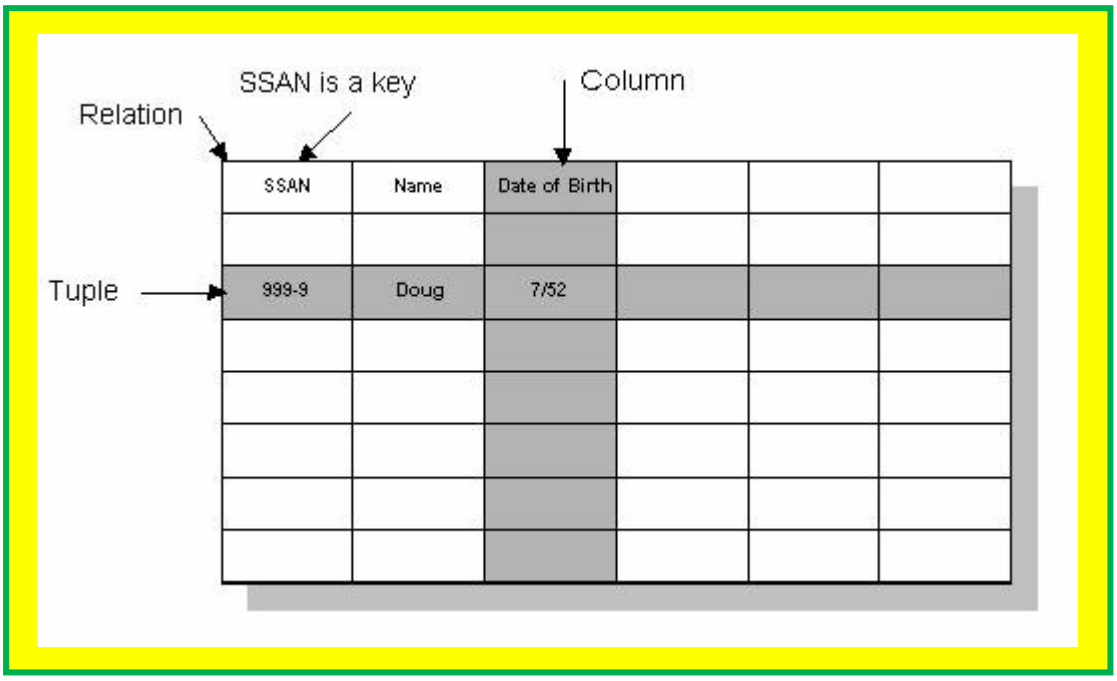

Figure 4.

A relationship is a logical connection between two or more entities. Meaningful data in an application includes relationships among its constituent parts. Relationships are essential to data modelling, yet the relational database model does not explicitly support relationships. Instead, primary keys, foreign keys, and referential integrity are used to implement some of the constraints implied by relationships.

In contrast, the Entity Data Model (EDM) provides explicit support for relationships in the data model, which results in flexible modelling capabilities. Relationship support extends to EDM queries, permitting explicit referencing and navigation based on relationships

Characteristics of Relationships

Relationships are characterized by degree, multiplicity, and direction. In data modelling scenarios, relationships have degree (unary, binary, ternary,

or n-ary), and multiplicity (one-to-one, one-to-many, or many-to-many). Direction can be significant in some associations, if, for example, the association is between entities of the same type.

The characteristics of relationships are shown in the following diagrams:

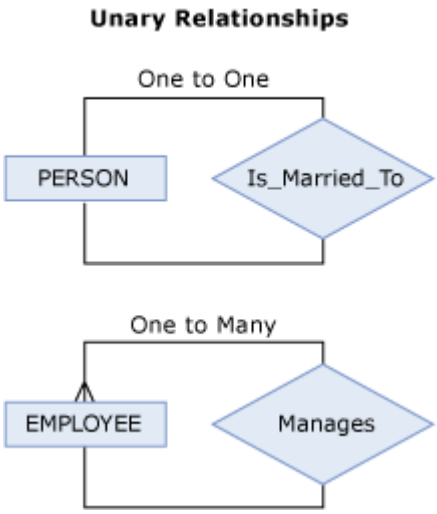

Figure 5a

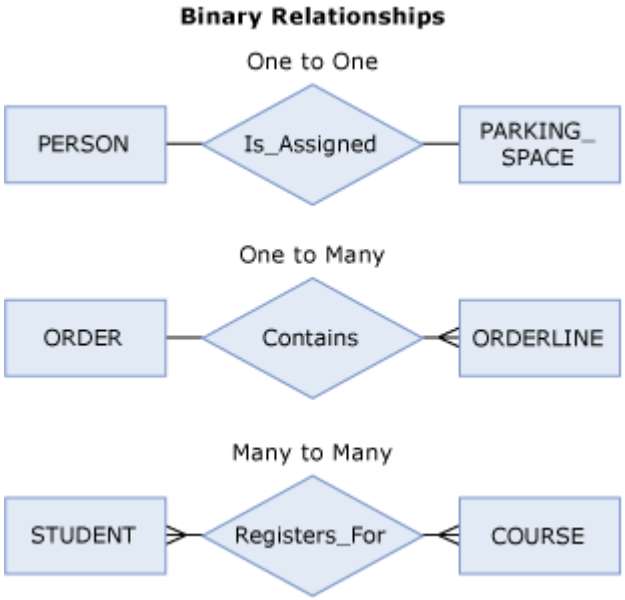

Figure 5b

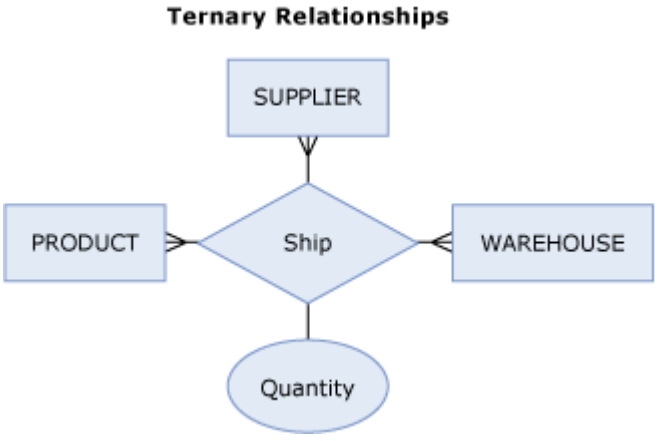

Figure 5c

The degree of the relationship in each diagram is represented by the number of rectangles. Relationships are represented by diamond-shaped figures. The lines between the diamonds and the rectangles represent the multiplicity of the relationships. A single line represents a one-to-one

relationship. A line that branches into three segments where it connects to the type represents the many ends of one-to-many or many-to-many relationships.

Degree

The degree of a relationship is the number of types among which the relationship exists. The most common degree of relationship is *binary*, which relates two types. In a unary relationship one instance of a type is related to another instance of the same type, such as the manager relationship between an employee and another employee. A *ternary* relationship relates three types and an *n-ary* relationship relates any number (n) of types. Ternary and n-ary relationships are mainly theoretical. The EDM supports unary and binary relationships.

Multiplicity

Multiplicity is the number of instances of a type that are related. A binary relationship exists between a book and its author, for example, where each book has at least one author. The relationship is specified between the class Book and the class Author, but the multiplicity of this relationship is not necessarily one book to one author. The multiplicity of the relationship indicates the number of authors a book has and the number of books each author has written. The degree of the relationship in this example is binary. The multiplicity of the relationship is many-to-many.

Direction

In the Entity Data Model (EDM), all relationships are inverse relations. An EDM association can be navigated starting from either end. If the entities at the ends of an association are both of the same type, the role attribute of the

EDM association End property can be used to specify directionality. An association between an employee and the employee's manager is semantically different from the two ends of the association. Both ends of the association are employees, but they have different Role attributes.

3.3 Concept of Attributes

The representation of the entity in the data model includes all of the characteristics and attributes of the entity, the actual data elements which must be present to fully describe each characteristic and attribute, and a representation of how that data must be grouped, organized and structured. Although the terms characteristics and attributes are sometimes be used interchangeably, attributes are the more general term, and characteristics are special use attributes.

An Attribute is any aspect, quality, characteristic or descriptor of either an entity or a relationship or may be a very abstract or general category of information, a specific attribute or element, or level of aggregation between these two extremes.
An attribute must also be

1. of interest to the corporation
2. capable of being described in real terms, and
3. relevant within the context of the specific environment of the firm.

An attribute must be capable of being defined in terms of words or numbers. That is, the attribute must have one or more data elements

associated with it. An attribute of an entity might be its name or its relationship to another entity. It may describe what the entity looks like, where it is located, how old it is, how much it weighs, etc. An attribute may describe why a relationship exists, how long it has existed, how long it will exist, or under what conditions it exists.

An attribute is an aspect or quality of an entity which describes it or its actions. An attribute may describe some physical aspect, such as size, weight or colour, or an aspect of the entity's location such as place of residence or place of birth. It may be a quality such as the level of a particular skill, educational degree achieved, or the dollar value of the items represented by an order.

A characteristic is some general grouping of data elements which serve to identify or otherwise distinguish or set apart one thing or group of things from another. A characteristic is a special form of attribute. It may be a very abstract or general category of information, an element, or level of aggregation between these two extremes. It is also some aspect of the entity that is required to gain a complete understanding of the entity, its general nature, its activities or its usage.

3.4 Entity Concept

An entity type is an abstraction that represents classes of real-world objects. An entity is a Person, Place, Plant, Thing, Event, or Concept of interest to the business or organization about which data is likely to be kept. For

example, in a school environment possible entities might be Student, Instructor, and Class. An entity type refers to a generic class of things such as Company and its property is described by its attribute types and relationship types.

An entity usually has attributes (i.e., data elements) that further describe it. Each attribute is a characteristic of the entity. An entity must possess a set of one or more attributes that uniquely identify it (called a primary key). The entities on an Entity-Relationship Diagram are represented by boxes (i.e., rectangles). The name of the entity is placed inside the box.

Identifying entities is the first step in Data Modelling. Start by gathering existing information about the organization. Use documentation that describes the information and functions of the subject area being analyzed, and interview subject matter specialists (i.e., end-users). Derive the preliminary entity-relationship diagram from the information gathered by identifying objects (i.e., entities) for which information is kept. Entities are easy to find. Look for the people, places, things, organizations, concepts, and events that an organization needs to capture, store, or retrieve.

There are three general categories of entities:

Physical entities are tangible and easily understood. They generally fall into one of the following categories:

- people, for example, doctor, patient, employee, customer.
- property, for example, equipment, land and buildings, furniture and fixtures, supplies.

- products, such as goods and services.

Conceptual entities are not tangible and are less easily understood. They are often defined in terms of other entity-types. They generally fall into one of the following categories:

- organizations, for example, corporation, church, government,
- agreements, for example, lease, warranty, mortgage,
- abstractions, such as strategy and blueprint.

Event/State entities are typically incidents that happen. They are very abstract and are often modelled in terms of other entity-types as an **associative entity**. Examples of events are purchase, negotiation, service call, and deposit. Examples of states are ownership, enrolment, and employment.

There are also three types of entities:

Fundamental Entities: These are entities that depict real things (Person, Place, or Concept, Thing etc).

Associative Entities: These are used for something that is created that joins two entities (for example, a receipt that exists when a customer and a salesperson complete a transaction).

Attributive Entities: These are used for data that is dependent upon a fundamental entity and are useful for describing attributes (for example, to identify a specific copy of a movie title when a video store has multiple copies of each movie).

3.5 Entity Relationship Model

Structured data is stored in databases. Along with various other constraints, this data's structure can be designed using entity relationship modelling, with the end result being an entity relationship diagram.

Data modelling entails the usage of a notation for the representation of data models. There are several notations for data modelling. The actual model is frequently called "Entity relationship model", because it depicts data in terms of the entities and relationships described in the data.

An entity-relationship model (ERM) is an abstract conceptual representation of structured data. Entity-relationship modelling is a relational schema database modelling method, used in software engineering to produce a type of conceptual data model (or semantic data model) of a system, often a relational database, and its requirements in a top-down fashion.

- The elements that make up a system are referred to as **entities**. A **relationship** is the association that describes the interaction between entities.
- An **entity-relationship diagram** is a graphical depiction of organizational system elements and the association among the elements. E-R diagrams can help define system boundaries.

An E-R diagram may also indicate the **cardinality** of a relationship. Cardinality is the number of instances of one entity that can, or must, be

associated with each instance of another entity. In general we may speak of one-to-one, one-to-many, or many-to-many relationships.

There are several different styles used to draw Entity-Relationship diagrams. The **Kendall** and **Kendall text** uses the Crow's Foot notation. Using this notation entities are represented by rectangles and relationships are indicated by lines connecting the rectangles. Cardinality is shown by a series of "tick marks" and "crows feet" superimposed on the relationship lines.

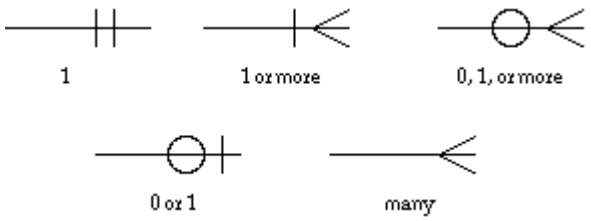

Figure 6

In the following example each student fills one seat in a class. Each seat is filled by one student. (In this usage a "seat" implies not only a physical place to sit but also a specific day and time.) This is a one-to-one relationship.

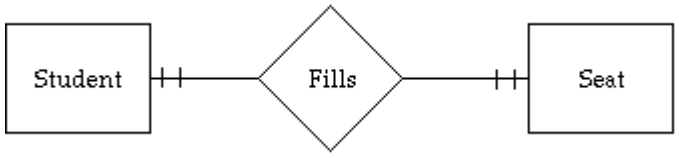

Figure 7

In the next example a single instructor may teach several courses. Each

course has only one instructor. This is a one-to-many relationship.

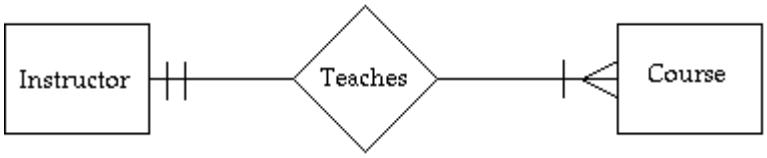

Figure 8

As shown below, a single student may register for several courses. A single course can have many students enrolled in it. This is the many-to-many relationship.

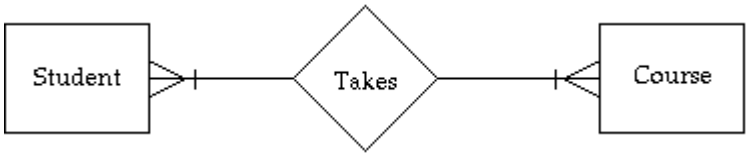

Figure 9

The next example shows a relationship in which it is possible that no instances exist. Each professor may teach several course sections but may not teach at all if on sabbatical. Assume there is no team teaching; therefore each section must have a single professor.

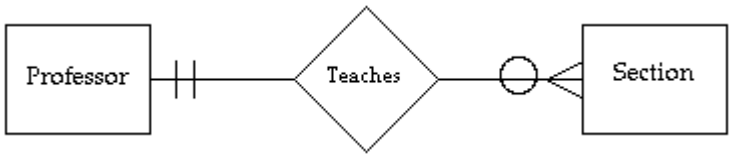

Figure 10

Finally, a more complex example which shows more than one relationship. All of the examples above depict single relationships. An actual E-R diagram would show the many entities and relationships that exist within a system. Here each department offers at least one course; there is no cross-

listing of courses with other departments. Each course must have at least one section but often has several sections.

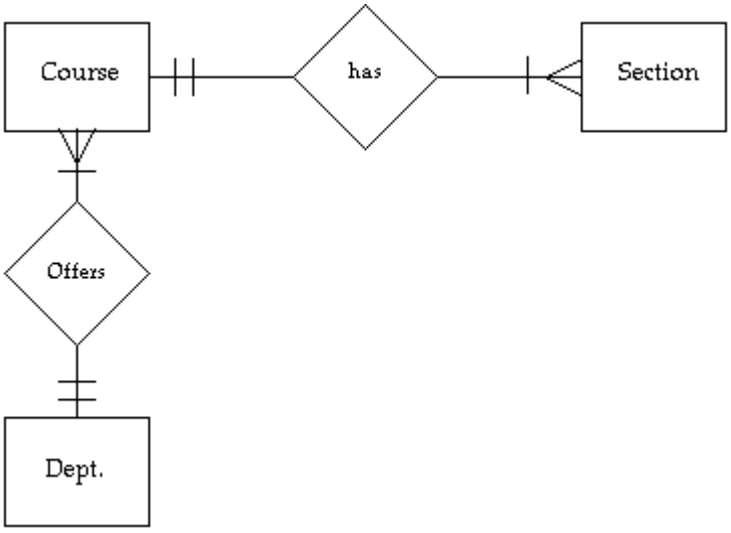

Figure 11

The E-R notation used in the Kendall and Kendall text (4th through 6th editions) also allows for distinguishing different types of entities. A plain rectangle is used for what is termed a **fundamental entity**, that is, an entity that is a real thing (person, place, or thing). The term **associative entity** is used for something that is created that joins two entities (for example, a receipt that exists when a customer and a salesperson complete a transaction). And, the term **attributive entity** is used for data that is dependent upon a fundamental entity and is useful for describing attributes (for example, to identify a specific copy of a movie title when a video store has multiple copies of each movie).

Figure 12

In an entity relationship model, attributes can be composite, derived, or multi-valued.

Multi-valued attributes might have more than one value in one or more instances of its entity. These attributes are denoted with a two line ellipse. So, if a piece of software happens to run on more than one operating system, it could have the attribute “platform”; this is a multi-valued attribute. Composite attributes, on the other hand, are those attributes that might contain two or more attributes.

A composite attribute will have contributing attributes of its own. An example of a composite attribute is an address, as it is composed of attributes that include street address, city, state/region, country, etc.

Finally, there are **derived attributes**. The value of these attributes is dependent wholly on another attribute. Derived attributes are denoted with dashed ellipses. So, in a database of employees that has an age attribute, the age attribute would have to be derived from an attribute of birth dates.

These models are being used in the first stage of information system design during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modelling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain universe of discourse i.e. area of interest.

3.6 Common Data Modelling Notations

The figure below presents a summary of the syntax of four common data modelling notations: Information Engineering (IE), Barker, IDEF1X, and the Unified Modeling Language (UML). This diagram isn't meant to be comprehensive; instead its goal is to provide a basic overview.

Comparing the syntax of common data modelling notations.

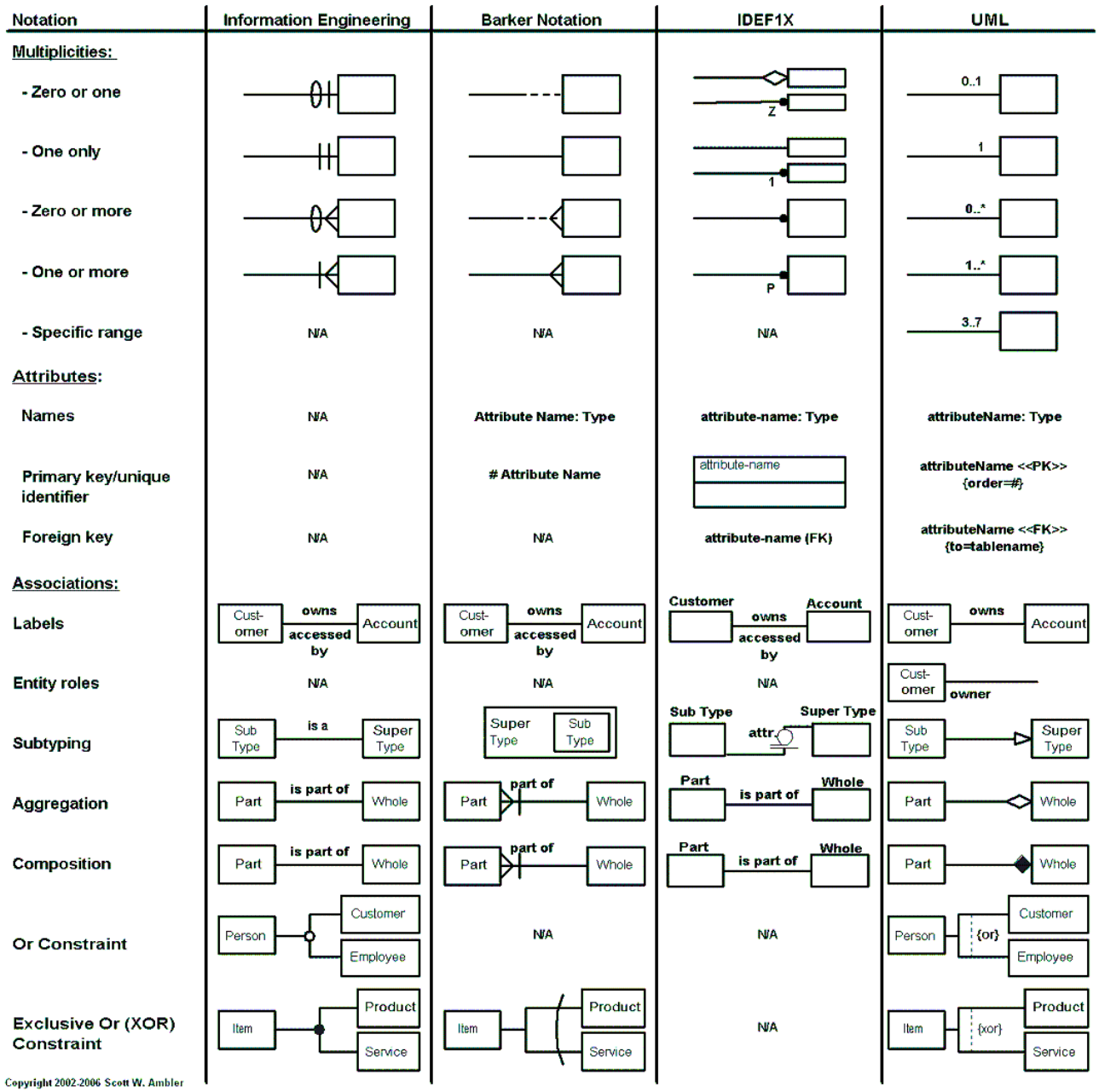

Figure 13.

4.0 Conclusion

The brief introduction to the concepts of data modelling helps individuals and organisations acquire knowledge of how physical data are diagrammatically modelled. And for this model be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure. The Entity-Relation Model (ER) is the most common method used to build data models for relational databases.

5.0 Summary

In this unit we have learnt that:

- ❖Generic data models define standardized general relation types, togeth er with the kinds of things that may be related by such a relation type.
- ❖The relational model used the basic concept of a relation or table, as a relationship is a logical connection between two or more entities, characterised by degree, multiplicity and direction.
- ❖ An Attribute is any aspect, quality, characteristic or descriptor of either an entity or a relationship, while an entity type is an abstraction that represents classes of real-world objects.
- ❖An entity-relationship model (ERM) is an abstract conceptual representation of structured data.

6.0 Tutor Marked Assignment

1. (a) What do you understand by the generalisation of conventional data mode?

   (b) Explain the concept of relational model in data modelling.

2. (a) Explain concept of entity in data modelling.

   (b) Briefly explain the concept of entity-relationship model.

7.0 Further Reading and Other Resources

Batini, C., S. Ceri, S. Kant, and B. Navathe. Conceptual Database Design: An Entity RelationalApproach. The Benjamin/Cummings Publishing Company, 1991.

Date, C. J. An Introduction to Database Systems, 5th ed. Addison-Wesley, 1990.

Fleming, Candace C. and Barbara von Halle. Handbook of Relational Database Design. Addison-Wesley, 1989.

Kroenke, David. Database Processing, 2nd ed. Science Research Associates, 1983.

Martin, James. Information Engineering. Prentice-Hall, 1989.

Reingruber, Michael C. and William W. Gregory. The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models. John Wiley & Sons, Inc., 1994.

Simsion, Graeme. Data Modeling Essentials: Analysis, Design, and Innovation. International Thompson Computer Press, 1994.

Teory, Toby J. Database Modeling & Design: The Basic Principles, 2nd ed.. Morgan Kaufmann Publishers, Inc., 1994.

J.H. ter Bekke (1991). *Semantic Data Modeling in Relational Environments*

Martin E. Modell (1992). *Data Analysis, Data Modeling, and Classification*.

Module 1 Concepts of Data Modelling

Unit 3 Data Models

1.0 Introduction

The goal of reading this section is not only for IT professionals to learn how to become a data modeller but also to gain an appreciation of what is involved in data models. And for an information system to be useful, reliable, adaptable, and economic, it must be based first on sound data modelling, and only secondarily on process analysis. So, every IT professional should be prepared to be involved in the creation of such models, be able to read an existing data models, understand when and when not to create a data model.

2.0 Objective

At the end of this unit, you should be able to:

- ❖ Explain what data model is all about
- ❖ Mention and explain the perspectives of data models
- ❖ Mention the features of a good data model
- ❖ Mention and explain the steps involved in the task of data modelling
- ❖ State the limitations of modelling data

3.0 Definition of Data Model

A data model is simply a diagram that describes the most important "things" in business environment from a data-centric point of view. For example, an **Entity Relationship Diagram** (ERD) below describes the

relationship between the data stored about products, and the data stored about the organizations that supply the products.

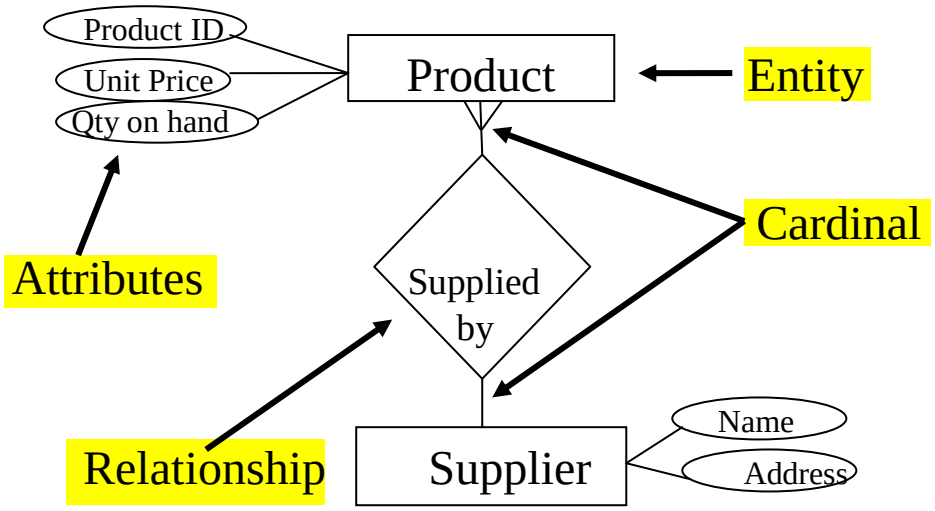

FIGURE 14: An ERD showing a relationship between products and suppliers.

The data requirements are recorded as a conceptual data model with associated data definitions. Actual implementation of the conceptual model is called a logical data model. To implement one conceptual data model may require multiple logical data models.

3.1 Perspectives of Data Models

A data model instance may be one of three kinds:

Conceptual schema: describes the semantics of a domain, being the scope of the model. For example, it may be a model of the interest area of an organization or industry. This consists of entity classes, representing kinds of things of significance in the domain, and relationships assertions about associations between pairs of entity

classes. A conceptual schema specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial 'language' with a scope that is limited by the scope of the model.

Logical schema: describes the structure of some domain of information. This consists of descriptions of tables and columns, object oriented classes, and XML tags, among other things.

Physical schema: describes the physical means by which data are stored. This is concerned with partitions, CPUs, Table spaces, and the like.

The significance of this approach is that, it allows the three perspectives to be relatively independent of each other. Storage technology can change without affecting either the logical or the conceptual model. The table/column structure can change without (necessarily) affecting the conceptual model. In each case, of course, the structures must remain consistent with the other model, as below:

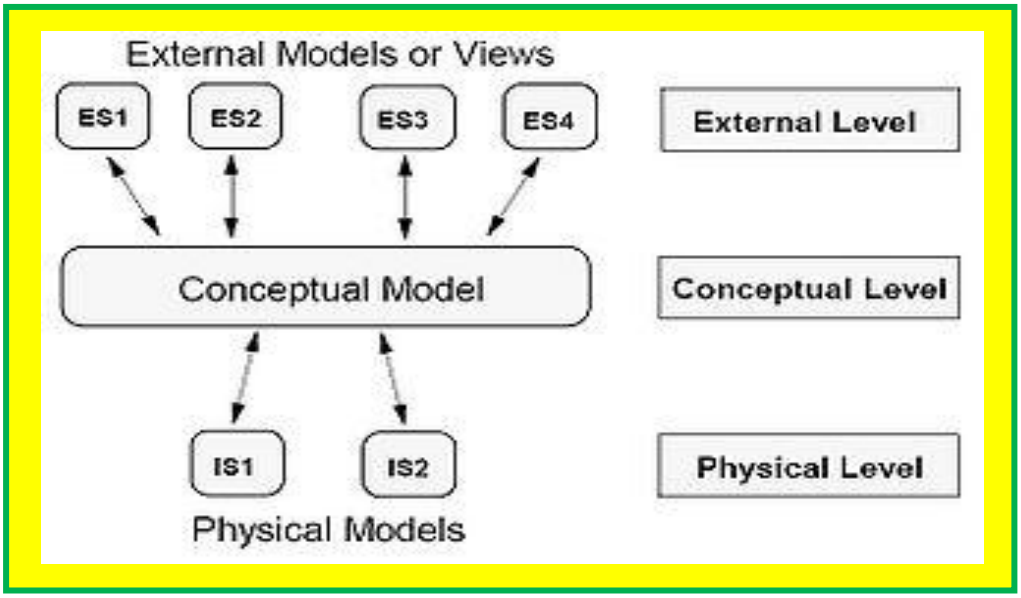

Figure 15

The three level architecture. This shows that a data model can be an external model (or view), a conceptual model, or a physical model. This is not the only way to look at

data models, but it is a useful way, particularly when comparing models.

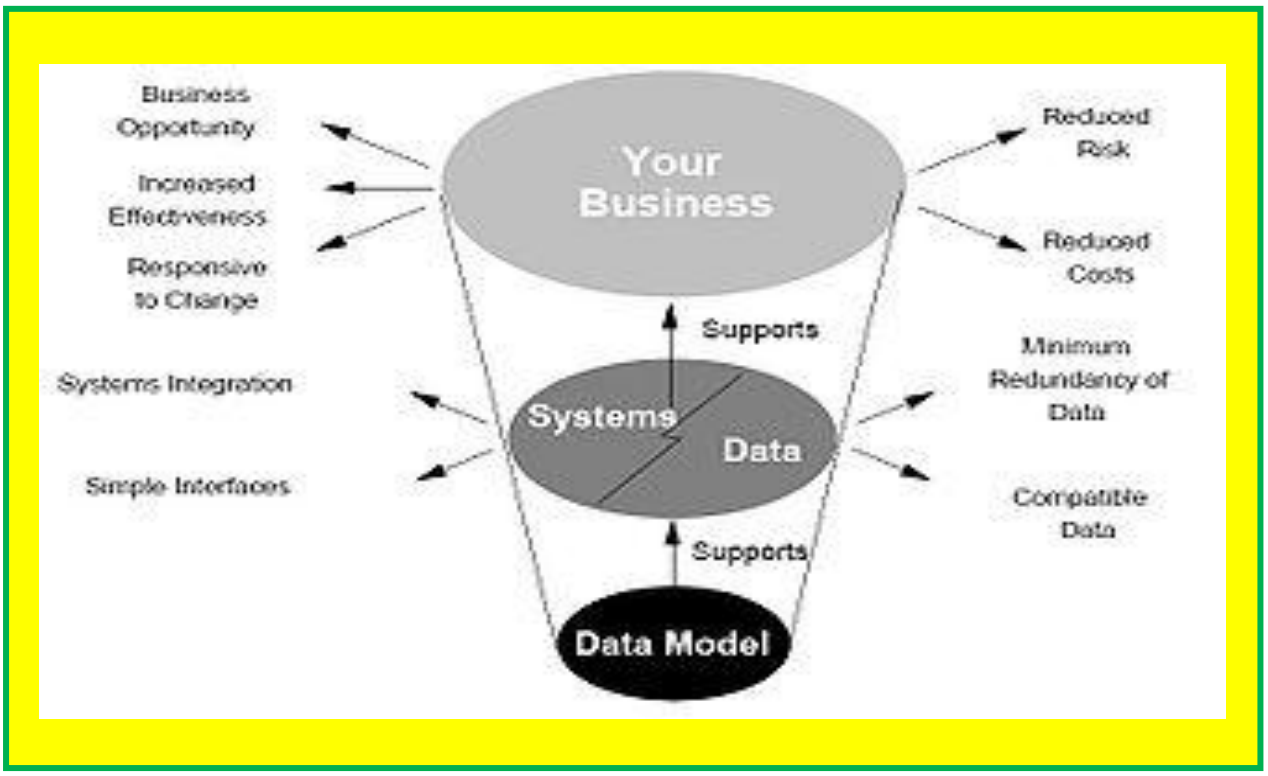

Figure 16 **How data models deliver benefit**

3.2 Features of a Good Data Model

The followings are what briefly make a good data model:

- Completeness
- Non-redundancy
- Enforcement of the Business rule
- Data reusability
- Stability and flexibility

- Communication and elegance
- Integration
- Conflicting objectives

3.3 How are Data Models Used in Practice?

- **Conceptual data models**. These models, sometimes called domain models, are typically used to explore domain concepts with project stakeholders. High-level conceptual models are often created as part of your initial requirements envisioning efforts as they are used to explore the high-level static business structures and concepts. On traditional teams conceptual data models are often created as the precursor to LDMs or as alternatives to LDMs.
- **Logical data models (LDMs)**. LDMs are used to explore the domain concepts, and their relationships, of your problem domain. This could be done for the scope of a single project or for your entire enterprise. LDMs depict the logical entity types, typically referred to simply as entity types, the data attributes describing those entities, and the relationships between the entities. LDMs are rarely used on Agile projects although often are on traditional projects (where they rarely seem to add much value in practice).
- **Physical data models (PDMs)**. PDMs are used to design the internal schema of a database, depicting the data tables, the data columns of those tables, and the relationships between the tables. PDMs often

prove to be useful on both Agile and traditional projects and as a result the focus of this article is on physical modelling.

Although LDMs and PDMs sound very similar, and they in fact are, the level of detail that they model can be significantly different. This is because the goal for each diagram is different – you can use an LDM to explore domain concepts with your stakeholders and the PDM to define your database design. Figure 17 presents a simple LDM and Figure 18 a simple PDM, both modelling the concept of customers and addresses as well as the relationship between them.

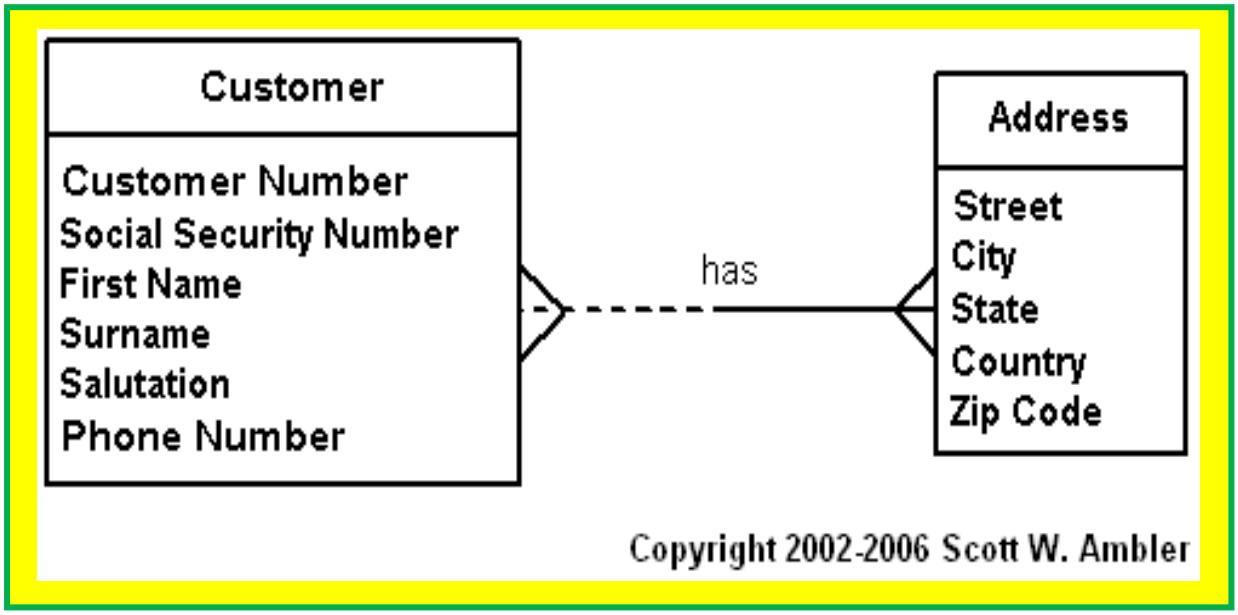

Figure 17. A simple logical data model.

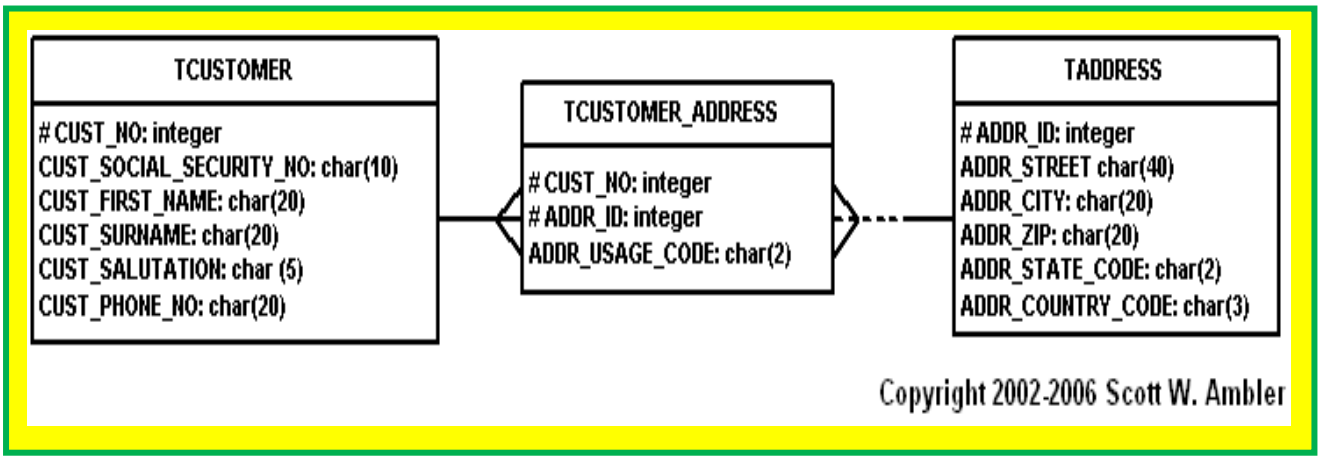

Figure 18. A simple physical data model.

3.4 How to Model Data

It is critical for an application developer to have a grasp of the fundamentals of data modelling so they can not only read data models but also work effectively with Databases who are responsible for the data-oriented aspects of your project.

The following tasks are performed in an iterative manner:

- Identify entity types
- Identify attributes
- Apply naming conventions
- Identify relationships
- Apply data model patterns
- Assign keys
- Normalize to reduce data redundancy
- Denormalize to improve performance

3.4.1 Identify Entity Types

An entity type, also simply called entity (not exactly accurate terminology, but very common in practice), is similar conceptually to object-orientation's concept of a class – an entity type represents a collection of similar objects. An entity type could represent a collection of people, places, things, events, or concepts. Examples of entities in an order entry system would include Customer, Address, Order, Item, and Tax. Ideally an entity should be normal data modelling world's version of cohesive. A normal entity depicts one concept, just like a cohesive class models one concept. For example, customer and order are clearly two different concepts; therefore it makes sense to model them as separate entities.

3.4.2 Identify Attributes

Each entity type will have one or more data attributes. For example, in Figure 17 you saw that the *Customer* entity has attributes such as *First Name* and *Surname* and in Figure 18 that the *TCUSTOMER* table had corresponding data columns *CUST_FIRST_NAME* and *CUST_SURNAME* (a column, is the implementation of a data attribute within a relational database).

Attributes should also be cohesive from the point of view of your domain, something that is often a judgment call. – in Figure 17 we decided that we wanted to model the fact that people had both first and last names instead of just a name (e.g. "Scott" and "Ambler" vs. "Scott Ambler") whereas we did not distinguish between the sections of zip code (e.g. 90210-1234-5678). Getting the level of detail right can have a significant impact on your

development and maintenance efforts. This is because, over-specifying an attribute (e.g. having three attributes for zip code when you only needed one) can result in overbuilding your system and hence you incur greater development and maintenance costs than you actually needed.

3.4.3 Apply Data Naming Conventions

Your organization should have standards and guidelines applicable to data modelling, something you should be able to obtain from your enterprise administrators (if they don't exist you should lobby to have some put in place). These guidelines should include naming conventions for both logical and physical modelling; the logical naming conventions should be focused on human readability whereas the physical naming conventions will reflect technical considerations.

You can clearly see that different naming conventions were applied in Figures 17 and 18. The basic idea is that developers should agree to and follow a common set of modelling standards on a software project. Just like there is value in following common coding conventions, clean code that follows your chosen coding guidelines is easier to understand and evolve than code that doesn't, there is similar value in following common modelling conventions.

3.4.4 Identify Relationships

In the real world entities have relationships with other entities. For example, customers PLACE orders, customers LIVE AT addresses, and line items ARE PART OF orders. Place, live at, and are part of, are all terms that define relationships between entities. The relationships between

entities are conceptually identical to the relationships (associations) between objects.

Figure 19 below depicts a partial LDM (Logical Data Model) for an online ordering system. The first thing to notice is the various styles applied to relationship names and roles – different relationships require different approaches. For example the relationship between '*Customer*' and '*Order'* has two names, (places and *is placed by)*, whereas the relationship between '*Customer'* and '*Address'* has one. In this example having a second name on the relationship, the idea being that you want to specify how to read the relationship in each direction is redundant – you're better off to find a clear wording for a single relationship name, decreasing the clutter on your diagram.

Similarly you will often find that by specifying the roles that an entity plays in a relationship will often negate the need to give the relationship a name (although some **CASE tools** may inadvertently force you to do this).

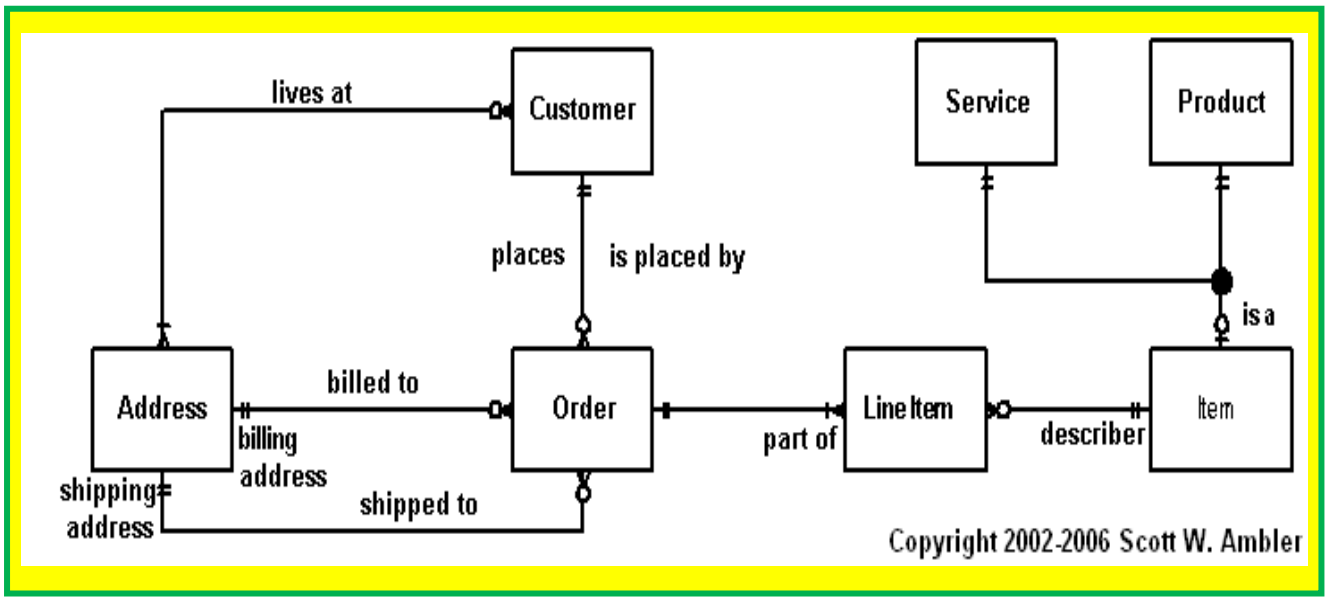

Figure 19. A logical data model (Information Engineering notation).

You also need to identify the cardinality and optionality of a relationship. Cardinality represents the concept of "how many" whereas optionality represents the concept of "whether you must have something." For example, it is not enough to know that customers place orders. How many orders can a customer place? None, one, or several? Furthermore, relationships are two-way streets: not only do customers place orders, but orders are placed by customers. This leads to questions like: how many customers can be enrolled in any given order and is it possible to have an order with no customer involved?

3.4.5 Apply Data Model Patterns

Some data modellers will apply common data model patterns, just as object-oriented developers will apply analysis patterns and design patterns. Data model patterns are conceptually closest to analysis patterns because they describe solutions to common domain issues.

3.4.6 Assign Keys

There are two fundamental strategies for assigning keys to tables:

First, you could assign a natural key which is one or more existing data attributes that are unique to the business concept. In the *Customer* table of Figure 20 there are two candidate keys, in this case *Customer Number* and *Social Security Number*.

Second, you could introduce a new column, called a surrogate key,

which is a key that has no business meaning. An example of which is the *Address ID* column of the *Address* table in Figure 20. Addresses don't have an "easy" natural key because you would need to use all of the columns of the *Address* table to form a key for itself (you might be able to get away with just the combination of *Street* and *Zip Code* depending on your problem domain), therefore introducing a surrogate key is a much better option in this case.

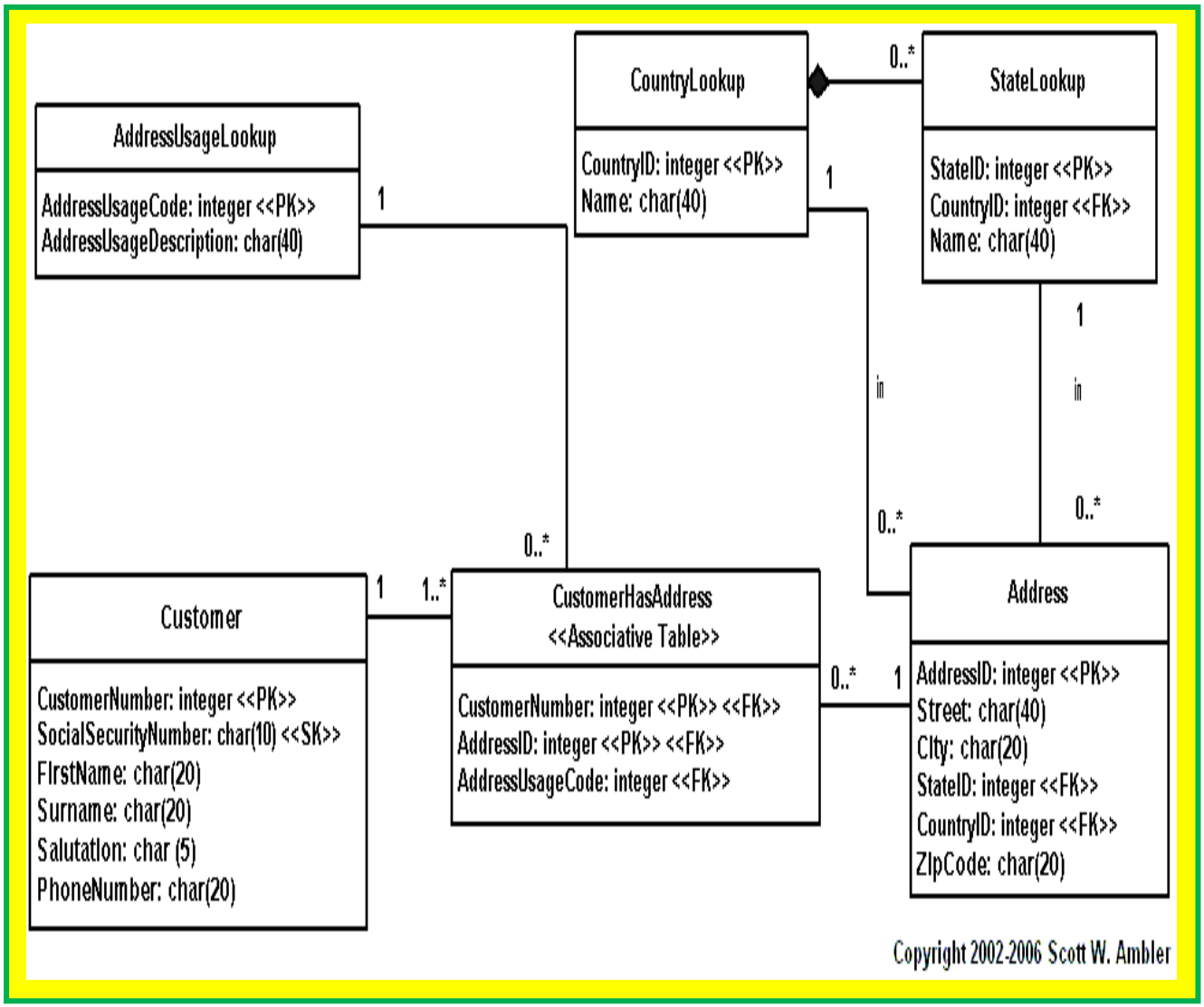

Figure 20. Customer and Address revisited (UML notation).

Let's consider Figure 20 in more detail. Figure 20 presents an alternative design to that presented in Figure 18, a different naming convention was adopted and the model itself is more extensive. In Figure 20 the *Customer* table has the *CustomerNumber* column as its primary key and *SocialSecurityNumber* as an alternate key. This indicates that the preferred way to access customer information is through the value of a person's customer number although your software can get at the same information if it has the person's social security number.

The *CustomerHasAddress* table has a composite primary key, the combination of *CustomerNumber* and *AddressID*. A foreign key is one or more attributes in an entity type that represents a key, either primary or secondary, in another entity type. Foreign keys are used to maintain relationships between rows. For example, the relationships between rows in the *CustomerHasAddress* table and the *Customer* table is maintained by the *CustomerNumber* column within the *CustomerHasAddress* table.

The interesting thing about the *CustomerNumber* column is the fact that it is part of the primary key for *CustomerHasAddress* as well as the foreign key to the *Customer* table. Similarly, the *AddressID* column is part of the primary key of *CustomerHasAddress* as well as a foreign key to the *Address* table to maintain the relationship with rows of *Address***.**

3.4.7 Normalize to Reduce Data Redundancy

Data normalization is a process in which data attributes within a data model are organized to increase the cohesion of entity types. In other

words, the goal of data normalization is to reduce and even eliminate data redundancy, an important consideration for application developers because it is incredibly difficult to stores objects in a relational database that maintains the same information in several places.

Table 1 summarizes the three most common normalization rules describing how to put entity types into a series of increasing levels of normalization. With respect to terminology, a data schema is considered to be at the level of normalization of its least normalized entity type. For example, if all of your entity types are at **second normal form** (2NF) or higher then we say that your data schema is at 2NF.

Level	Rule
First normal form (1NF)	An entity type is in 1NF when it contains no repeating groups of data.
Second normal form (2NF)	An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.
Third normal form (3NF)	An entity type is in 3NF when it is in 2NF and when all of its attributes are directly dependent on the primary key.

Table 1. Data Normalization Rules.

The next question is, why data normalization? The advantages of having a highly normalized data schema are that:

Information is stored in one place and one place only, reducing the possibility of inconsistent data.

Furthermore, highly-normalized data schemas in general are closer conceptually to object-oriented schemas because the object-oriented goals of promoting high cohesion and loose coupling between classes results in similar solutions (at least from a data point of view). This generally makes

it easier to map your objects to your data schema.

Unfortunately, normalization usually comes at a performance cost.

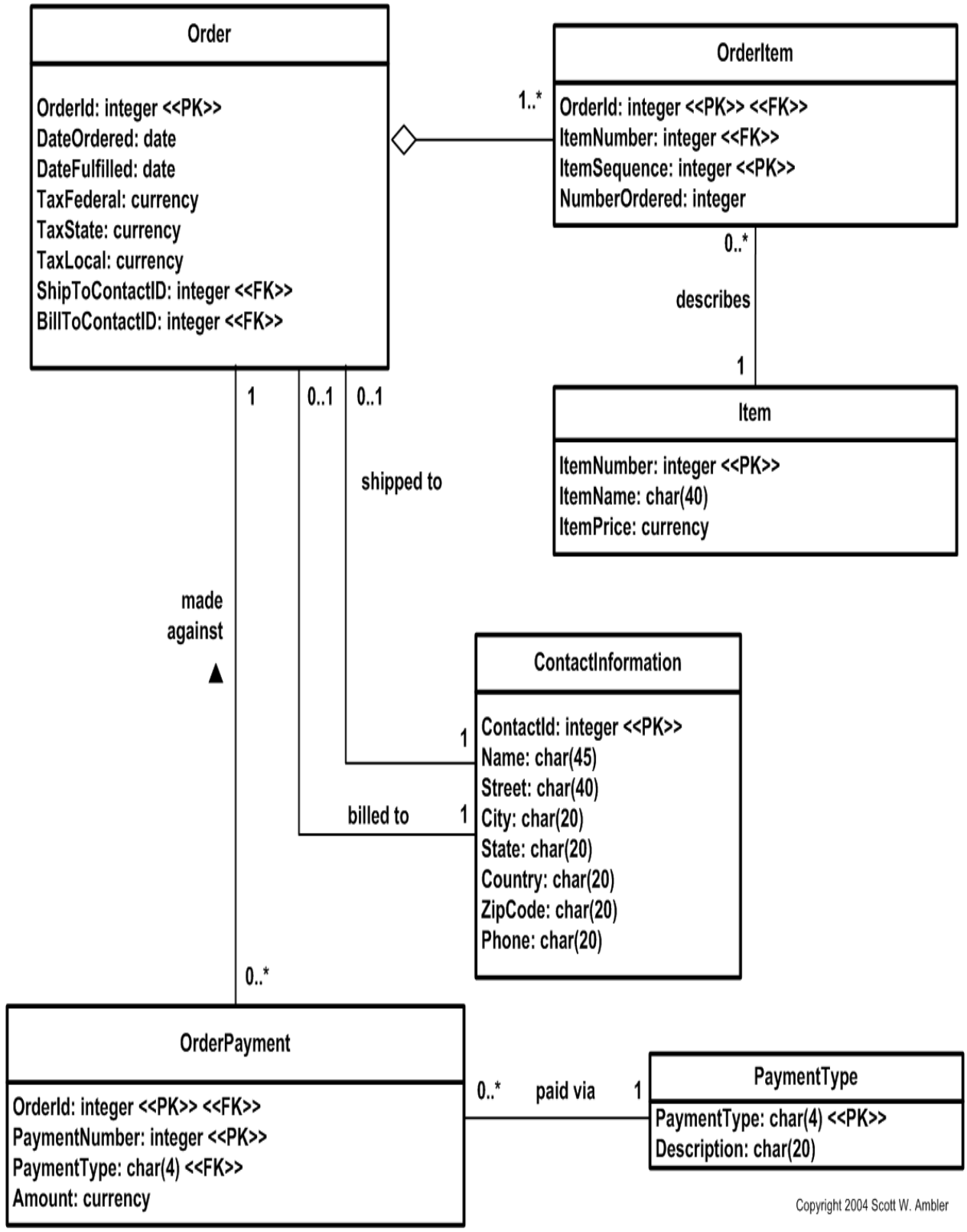

Figure 21. A normalized schema in 3NF (UML Notation).

3.4.8 Denormalize to Improve Performance

Normalized data schemas, when put into production, often suffer from performance problems. This makes sense – the rules of data normalization focus on reducing data redundancy, not on improving performance of data access. An important part of data modelling is to denormalize portions of your data schema to improve database access times. For example, the data model of Figure 22 looks nothing like the normalized schema of Figure 21.

To understand why the differences between the schemas exist, you must consider the performance needs of the application. The primary goal of this system is to process new orders from online customers as quickly as possible. To do this customers need to be able to search for items and add them to their order quickly, remove items from their order if need be, then have their final order totalled and recorded quickly. The secondary goal of the system is to the process, ship, and bills the orders afterwards.

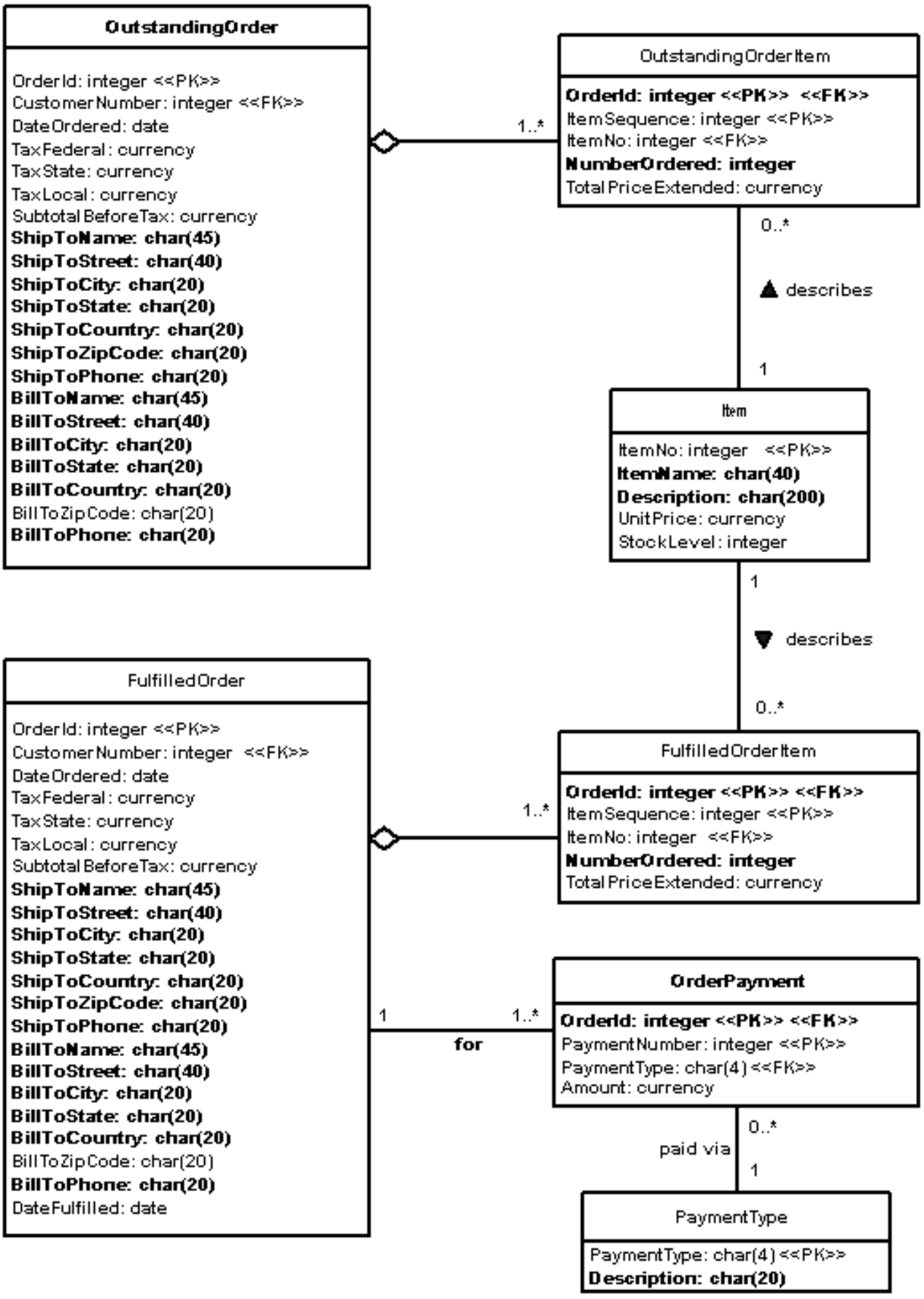

Figure 22. A Denormalized Order Data Schema (UML notation).

To denormalize the data schema the following decisions were made:

1. To support quick searching of item information the *Item* table was left alone.
2. To support the addition and removal of order items to an order the concept of an *OrderItem* table was kept, albeit split in two to support outstanding orders and fulfilled orders. New order items can easily be inserted into the *OutstandingOrderItem* table, or removed from it, as needed.
3. To support order processing the *Order* and *OrderItem* tables were reworked into pairs to handle outstanding and fulfilled orders respectively. Basic order information is first stored in the *OutstandingOrder* and *OutstandingOrderItem* tables and then when the order has been shipped and paid for the data is then removed from those tables and copied into the *FulfilledOrder* and *FulfilledOrderItem* tables respectively. Data access time to the two tables for outstanding orders is reduced because only the active orders are being stored there. On average an order may be outstanding for a couple of days, whereas for financial reporting reasons may be stored in the fulfilled order tables for several years until archived. There is a performance penalty under this scheme because of the need to delete outstanding orders and then resave them as fulfilled orders, clearly something that would need to be processed as a transaction.
4. The contact information for the person(s) the order is being shipped and billed to was also denormalized back into the *Order* table, reducing the time it takes to write an order to the database because

there is now one write instead of two or three. The retrieval and deletion times for that data would also be similarly improved.

Note that if your initial normalized data design meets the performance needs of your application, then it is fine. Denormalization should be resorted to only when performance testing shows that you have a problem with your objects and subsequent profiling reveals that you need to improve database access time.

How to Become Better At Modelling Data

How do you improve your data modelling skills? Practice, practice, practice.

Similarly you should take the opportunity to work with the enterprise architects within your organization.

You also need to do some reading.

3.5 Limitations of Data Models

Data models support data and computer systems by providing the definition and format of data. If this is done consistently across systems, then compatibility of data can be achieved. If the same data structures are used to store and access data, then different applications can share data. However, systems and interfaces often cost more than they should, to build, operate, and maintain.

They may also constrain the business rather than support it. The reason for these problems is a lack of standards that will ensure that, data models will both meet business needs and be consistent. Therefore, the major cause is that, the quality of the data models implemented in systems and interfaces is poor, with the following resultant effects:

- Business rules, specific to how things are done in a particular place, are often fixed in the structure of a data model. This means that small

changes in the way business is conducted lead to large changes in computer systems and interfaces.

- Entity types are often not identified, or incorrectly identified. This can lead to replication of data, data structure, and functionality, together with the attendant costs of that duplication in development and maintenance.
- Data models for different systems are arbitrarily different. The result of this is that complex interfaces are required between systems that share data. These interfaces can account for between 25-70% of the cost of current systems.
- Data cannot be shared electronically with customers and suppliers, because the structure and meaning of data has not been standardised. For example, engineering design data and drawings for process plant are still sometimes exchanged on paper.

4.0 Conclusion

The bulk of work in this unit is on how to model data and its exploration gives an insight into how structured data are stored in the data management systems such as relational databases, as it is known that, the main aim of data models is to support the development of information systems by providing the definition and format of data.

5.0 Summary

In this unit we have learnt that:

❖ A data model is simply a diagram that describes the most important

"things" in business environment from a data-centric point of view.

- ❖ The perspective of data models are conceptual, logical and physical schemas.
- ❖ What makes a good data includes Completeness, Non-redundancy, Enforcement of the Business rule, Data reusability, Stability and flexibility, Communication and elegance, Integration, etc.
- ❖ Eight basic steps are involved in how to model data.

6.0 Tutor Marked Assignment

1. (a) What do you understand by the term data model?
   (b) Differentiate between the perspective and the practical use of data model.

2. (a) Mention the features of a good data model
   (b) How do you model data?

7.0 Further Reading and Other Resources

David C. Hay (1996). *Data Model Patterns: Conventions of Thought*. New York: Dorset House Publishers, Inc.

Matthew West and Julian Fowler (1999). Developing High Quality Data Models. The European Process Industries STEP Technical Liaison Executive (EPISTLE).

Len Silverston (2001). *The Data Model Resource Book* Volume 1/2. John Wiley & Sons.

RFC 3444 - On the Difference between Information Models and Data Models

Len Silverston & Paul Agnew (2008). *The Data Model Resource Book: Universal Patterns for data Modelling* Volume 3. John Wiley & Sons.

Steve Hoberman, Donna Burbank, & Chris Bradley (2009). Data Modeling for the Business. Technics Publications, LLC

Andy Graham (2010), *The Enterprise Data Model: a framework for enterprise data architecture*

Michael R. McCaleb (1999). "A Conceptual Data Model of Datum Systems". National Institute of Standards and Technology. August 1999.

Matthew West and Julian Fowler (1999). Developing High Quality Data Models. The European Process Industries STEP Technical Liaison Executive (EPISTLE).

American National Standards Institute. 1975. *ANSI/X3/SPARC Study Group on Data Base Management Systems; Interim Report*. FDT (Bulletin of ACM SIGMOD) 7:2.

"Data Modelling for the Business", Steve Hoberman, Donna Burbank, Chris Bradley, Technics Publications, LLC 2009

Young, J. W., and Kent, H. K. (1958). "Abstract Formulation of Data Processing Problems". In: *Journal of Industrial Engineering*. Nov-Dec 1958. 9(6), pp. 471-479

Janis A. Bubenko jr (2007) "From Information Algebra to Enterprise Modelling and Ontologies - a Historical Perspective on Modelling for Information Systems". In: *Conceptual Modelling in Information Systems Engineering*. John Krogstie et al. eds. pp 1-18

Module 2 Semantic Data Modelling

Unit 1 Overview of Semantic Data Modelling

1.0 Introduction

In the previous Units, data modelling and models, as well as their concepts and developments, were explicitly examined. In this Unit, Semantic Data Modelling will be dealt with. And for this purpose, we will examine a new data modelling approach based on semantic principles, which results in inherently specified data structures, as a singular word or data item hardly can convey meaning to humans, but in combination with the context, a word gets more meaning.

Also, the logical data structure of a DBMS(Data Base Management System), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modelling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data.

Then, in a database environment, the context of data items is mainly defined by structure: a data item or object can have some properties ("horizontal structure"), but can also have relationships ("vertical structure") with other objects. In the relational approach vertical structure is defined by explicit referential constraints, but in the semantic approach structure is defined in an inherent way: a property itself may coincide with a reference to another object. This has important consequences for the semantic data manipulation language.

2.0 Objectives

At the end of this unit, you should be able to:

- ❖ Define semantic data modelling
- ❖ State and explain the principles of semantic data modelling
- ❖ Explain data integrity rule
- ❖ Explain additional data restrictions and declarative data derivations

3.0 Definition of Semantic Data Modelling

Semantic Data Modelling is a technique used to define the meaning of data within the context of its interrelationships with other data. In terms of resources, ideas, events, etc., the real world is symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

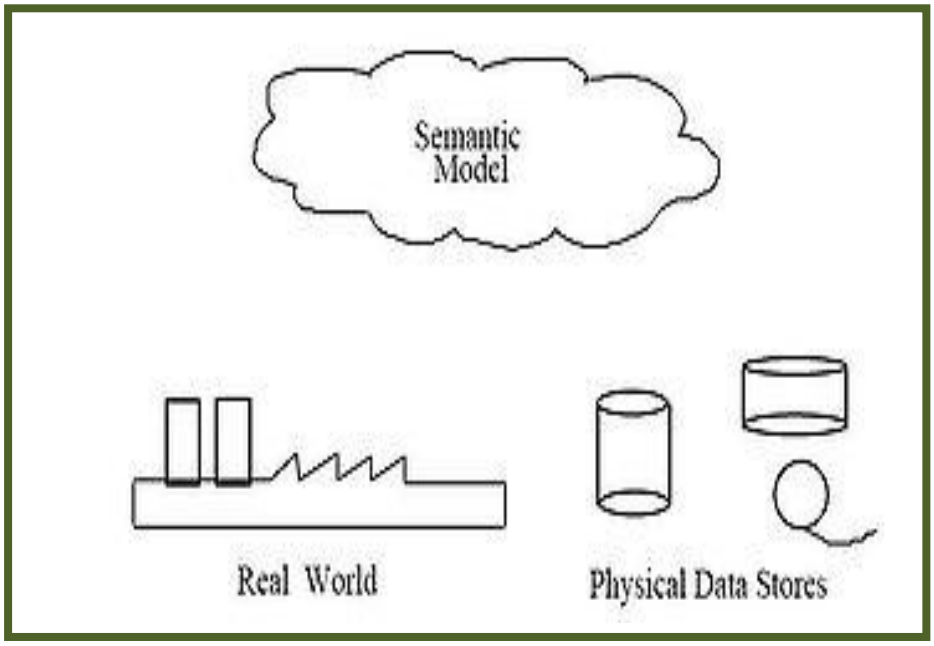

Figure 23 **Semantic data models**

3.1 Principles of Semantic Data Modelling

The objective of data modelling is to design a data structure for a database fitting as good as possible with some relevant world, often related to an organization with some information need. In general there is some relationship between a data model and a part of the existing world, but it is also possible that a data model has a relationship with some imaginary and abstract world. According to Smith and Smith (1977), three abstractions are very important for data modelling:

- Classification
- Aggregation
- Generalization

Classification is used to model instance of relations, aggregation to model has a relations and generalization to model is a relations. In semantic data modelling all three abstractions lead to a type definition (which can be base or composite). The semantic data model is, as contrasted with many other data models, based on only one fundamental notion: the type concept.

It is interesting to know that also the Xplain meta data model requires all three abstractions. Many other data modelling techniques (such as relational model) do not support these three abstractions and therefore are limited in modelling capabilities.

A semantic data model can be represented graphically in an Abstraction Hierarchy diagram showing the types (as boxes) and their inter-relations (as

lines). It is hierarchical in the sense that the types which reference other types are always listed above the referenced type. This simple notation principle makes the diagrams very easy to read and understand, even for non-data modellers.

Aggregation: A type can have attributes (properties), but attributes have a type of their own (for example the type “name”) and must be part of a composite type: thus an attribute defines a connection or aggregation of two types. For example, “name” can be an attribute of the type “employee”. Using the semantic language, this attribute can be specified as “employee its name”. Each composite type has some positive number of attributes, but a **base type** such as “name” does not have any attribute. Within a data model, a type can be identified by a name. At the same time a collection of properties (attributes) identifies a composite type: there may not be two types with the same collection of attributes.

Moreover, within a data model a composite type has only one definition: the principle of **convertibility**. Convertibility not only applies to the conceptual level (the data model), but also applies to instances of composite types. Using the following simplified definition of the type “employee”, the two mentioned instances of “employee” may not be registered in a same database:

type employee = name, address, town, department.

employee	**name**	**address**	**town**	**Department**
123	John	43 Q Street	Greenfield	12
432	John	43 Q Street	Greenfield	12

These two instances are conflicting with the principle of convertibility, but are allowed by an equivalent relational model because relational modelling only requires uniqueness of a primary key value. However, readers might not agree, because a father and his son can have the same shown properties. If we want to deal with such possible situations then it is necessary to extend the definition “employee” with properties such as “birth_date” and “function” or “salary” enabling us to make a distinction between the two instances.

Types are described by their **permanent** properties, in particular the properties relevant for the information needs at hand. If a property or attribute is not relevant to all instances of a certain composite type then the data definition must be improved. The definition of object types related to a same real life object as “person” needs not be the same for all organizations; the FBI and CIA have more information needs than a local administration department.

Generalization: In addition to aggregation as a building principle for data modelling, generalization is another building stone. An example is that a student administration can be interested in different kinds of students: students with or without a job. We could model this as follows:

type student = name, address, town, birth_date, faculty, employer.

The problem with this solution is that in the case of nil employer the attribute “student its employer” is not relevant at all. If this model is implemented the people of the student administration department cannot be sure about the relevance of the attribute “student its employer”. If they have

to insert new instances of “student” the data model as such does not tell them what to do. They can make two kinds of errors: they can ignore the field for employer even if it still is relevant, or they can refer to some employer if it is not relevant. Therefore it is better to design a data model where the type “student” has all the properties that are relevant to all students, which implies that there is not an attribute “student its employer”. We call this **generalization**: the cross section of properties of “student” and “working student” defines the type “student” and the type “working student” is a **specialization** of “student”: it has the additional property “working student its employer”:

type student = name, address, town, birthdates, faculty.
type working student = [student], employer.

The square brackets specify that for each instance of “student” there is at most one instance of “working student” and there is one instance of “student” referred to by an instance of “working student”. Using generalization/specialization it is clear to users which attribute fields are mandatory for registration. Now there is no doubt about the relevance of attribute fields in an application: each attribute field must be filled correctly and NULL-values are not allowed! The absence of NULL-values is one of the requirements for applying a semantic data manipulation language.

3.2 Data Integrity Rules

Data model specifications imply validity of certain integrity rules. Two inherent integrity rules are recognized for type definitions in a semantic data model:

- **Relatability:** Each attribute in a type definition is related to one and only one equally named type, while each type may correspond with various attributes in other types.
- **Convertibility:** Each type definition is unique: there are no type definitions carrying the same name or the same collection of attributes.

It is important to realize that these two integrity rules require neither separate specification nor declaration by procedures; they are inherent to the type definitions in a semantic data model.

3.3 Additional Data Restrictions

In addition to restrictions inherent from the data model there is often need for additional more complex restrictions on data states that cannot be specified in a data model itself. These additional restrictions can be specified as :

Static restrictions (applicable in all states) are restrictions that apply always. These are defined as assertions. An **assertion** specifies a derivable attribute or a derivable single variable, possibly some value restriction, and the calculation of the derivable item(s). An example is that in many organizations there is only one manager per department:

assert department its managercount (1..1) = count employee where function = manager per department.

If an organization wants to guard that the number of employees is not more than 100, the following assertion using a single variable can be specified:

assert maxemployees (0..100) = count employee.

Dynamic restrictions (applicable only in certain states, such as during creation or after update) have to deal with state transitions of data.

3.4 Declarative Data Derivations

The assert command - as explained in static restriction - can be used to specify derivable attributes. This is extremely useful for modelling complex data derivations, such as needed for user applications (e.g. total order amount), reports (e.g. grouping per x, Year-To-Date and totals) and data analysis (top 10 products).

An assert command derives only one attribute at a time. For complex derivations you need to define multiple assertions building on each other. This principle ensures modularity (thus easy to understand, test and maintain) and reusability (of intermediate derived data) for other queries.

4.0 Conclusion

With semantic data modelling as the meaning of data within the context of its interrelationships with other data. So, its knowledge will help individuals in having understanding of how data are ethically modelled, for the benefit of an enterprise.

5.0 Summary

In this unit we have learnt that:

- ❖ Semantic Data Modelling is a technique used to define the meaning of data within the context of its interrelationships with other data.
- ❖ The principles of semantic data modelling include Classification, aggregation, and generalisation.

- ❖ The data model specifications imply validity of certain integrity rules such as relatability and convertibility.
- ❖ The additional restrictions can be specified as; static and dynamic restrictions.

6.0 Tutor Marked Assignment

1. (a) Define semantic data modelling
   (b) Explain the principles of semantic data modelling
2. Explain static and dynamic data restrictions.

7.0 Further Reading and Other Resources

Jack, Colin, and Colin Keillor. "Semantic Data Models." Unpublished essay. School of Computing, Napier University. 15 Oct. 2006 <http://www.soc.napier.ac.uk/module.php3

Semantic Data Model. 13 Aug. 2001. COMTECO Ltd. 15 Oct. 2006 <http://www.comteco.ru/EN/DATAMODE/datamode.htm

Object Role Modeling: An Overview (msdn.microsoft.com). Retrieved 19 September 2008.

Grady Booch, Ivar Jacobson & Jim Rumbaugh (2000) OMG Unified Modeling Language Specification, Version 1.3 First Edition: March 2000. Retrieved 12 August 2008.

Halpin, T.A. 2001a, *Information Modeling and Relational Databases*, Morgan Kaufmann Publishers, San Francisco (www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672-6).

Len Silverton (2001). *The Data Model Resource Book* Volume 1/2. John Wiley & Sons.

Module 2 Semantic Data Modelling

Unit 2 Semantic Data Models

1.0 Introduction

According to Klas and Schrefl (1995), the "overall goal of semantic data models is to capture more meaning of data, by integrating relational concepts with more powerful abstraction concepts known from the Artificial Intelligence field. The idea is to provide high level modelling primitives as integral part of a data model, in order to facilitate the representation of real world situations.

2.0 Objective

At the end of this unit, you should be able to:

- ❖ Explain what semantic data model is all about.
- ❖ State the basic requirement of semantic data model
- ❖ Mention the areas of application of semantic data model
- ❖ Develop semantic binary object-oriented data model.

3.0 Overview of Semantic Data Model

The logical data structure of a database management system (DBMS), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data, because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modelling techniques. That is,

techniques to define the meaning of data within the context of its interrelationships with other data.

The real world, in terms of resources, ideas, events, etc., is symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

3.1 Definition of Semantic Data Models

A semantic data model in software engineering has various meanings:

It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances. Such a semantic data model is an abstraction that defines how the stored symbols (the instance data) relate to the real world.

It is a conceptual data model that includes the capability to express information that enables parties to the information exchange, to interpret meaning (semantics) from the instances, without the need to know the meta-model. Such semantic models are fact oriented (as opposed to object oriented). Facts are typically expressed by binary relations between data elements, whereas higher order relations are expressed as collections of binary relations.

Typically binary relations have the form of triples: Object-RelationType-Object. For example: the Eiffel Tower <is located in> Paris. Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the instances, it is required that the

meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardize such relation types. This means that the second kind of semantic data models enable that, the instances express facts that include their own meaning.

The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies that in general they have a wider applicability than relational or object oriented databases.

3.2 History of Semantic Data Models

The need for semantic data models was first recognized by the U.S. Air Force in the mid-1970s as a result of the Integrated Computer-Aided Manufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM Program identified a need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, the ICAM Program developed a series of techniques known as the IDEF (ICAM DEFinition) Methods which included the following:

IDEF0 used to produce a “function model” which is a structured representation of the activities or processes within the environment or system.

IDEF1 used to produce an “information model” which represents the structure and semantics of information within the environment or system.
IDEF1X is a semantic data modelling technique. It is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases.
IDEF2 used to produce a “dynamics model” which represents the time varying behavioural characteristics of the environment or system.

3.3 Semantic Data Model Requirements

The data model should meet the following requirements**:**

- Allow to specify well-defined schemata (schema definition language).
- Support dynamic schema evolution to capture new or evolving types of semantic information.
- Be simple to use, light-weight, make no assumptions about the semantics of the metadata.
- Be platform independent and provide interoperability between applications that manage and exchange metadata.
- Facilitate integration with resources outside the file store and support exporting metadata to the web.
- Leverage existing standards and corresponding tools, such as query languages.

3.4 Applications of Semantic Data Models

A semantic data model can be used to serve many purposes, including:

- Planning of Data Resources: A preliminary data model can be used to provide an overall view of the data required to run an enterprise. The model can then be analyzed to identify and scope projects to build shared data resources.
- Building of Shareable Databases: A fully developed model can be used to define an application independent view of data which can be validated by users and then transformed into a physical database design for any of the various DBMS technologies. In addition to generating databases which are consistent and shareable, development costs can be drastically reduced through data modelling.
- Evaluation of Vendor Software: Since a data model actually represents the infrastructure of an organization, vendor software can be evaluated against a company's data model in order to identify possible inconsistencies between the infrastructure implied by the software and the way the company actually does business.
- Integration of Existing Databases: By defining the contents of existing databases with semantic data models, an integrated data definition can be derived. With the proper technology, the resulting conceptual schema can be used to control transaction processing in a distributed database environment. The U.S. Air Force Integrated Information Support System (I2S2) is an experimental development and demonstration of this type of technology applied to a heterogeneous DBMS environment.

3.5 The Semantic Binary Object-Oriented Data Model

This section introduces SBOODM, which is an example of an object-oriented data model. SBOODM has been defined with the intention to expose the basics of the object-oriented paradigm to the readers already familiar with semantic modelling. It is a semantic binary model augmented with an abstraction mechanism to model the behaviour of objects.

Object-oriented schemas

An object-oriented schema must capture not only the structural properties of an application's real world but also its behavioural properties. In the SBOODM, a schema lists all aspects of an application's real world in graphical form and in the appendix. More formally a SBOODM schema is defined as follows:

SBOODM schema

A semantic binary schema whose appendix lists the methods defined for each of the categories and the interface to each category. The concepts of *method* and *category interface* are defined in the sections that follow.

Methods:

In the object-oriented paradigm it is possible to predefine a library of operations (procedures and functions) for a given schema. These operations are called **methods** and are the primary means for modelling the behavioural characteristics of an application's real world. Methods can be constructed by data manipulation primitives. Of special interest are the object methods and category methods. In the SBOODM methods are

implemented via Extended Pascal procedures or functions and are defined as follow:

Object method: This is an extended Pascal **function** or **procedure** which satisfies the following:

- The first formal parameter of the method is of type *ABSTRACT* and must be a value parameter. When a method *m* is invoked and an object *o* is assigned to its first parameter, then we say that *m* **is invoked on** *o*.
- The method restricts the first argument to belong to a particular category. When that is a category *C*, we say that the method **is defined on** *C*.

We allow for the following convenient syntactic abbreviation for the header of a procedure method *m* defined on a category *C*:

procedure *m* (first-argument:*C*; other-arguments); (and similarly for function methods.)

Example
The method

```
procedure print-name(i:INSTRUCTOR);
begin
writeln(i.LAST-NAME, i.FIRST-NAME)
end
```

can be regarded as a syntactic abbreviation for

```
procedure print-name(i:ABSTRACT);
begin
if not (i is an INSTRUCTOR) then
writeln(`Error: The input object is not of the category INSTRUCTOR')
else writeln(i.LAST-NAME, i.FIRST-NAME)
end
```

Example.

An object method which computes and returns the age of a person.

function get-approximate-age(person: *PERSON*): *INTEGER*;
var current-year: *INTEGER*;
begin
read(current-year);
get-age := current-year - person.*BIRTH-YEAR*;
end;

Example

An object method which enrolls a student in a course.

procedure enroll(student-to-enroll: *STUDENT*; course-name: *STRING*;
quarter: *STRING*; year: *INTEGER*);
var offer, enrollment : *ABSTRACT*;
begin transaction for offer **in** *COURSE-OFFERING* **where** offer .
THE-COURSE.NAME = course-name **and** offer.*THE-QUARTER.YEAR* = year
and offer.
THE-QUARTER.SEASON = quarter **do begin create new** enrollment **in**
COURSE-ENROLLMENT;
enrollment.*THE-STUDENT* := student-to-enroll;
enrollment.*THE-OFFER* := offer;
end
end;

Example .

An example of an object method which interactively registers a student into up to five courses.

procedure register(student-to-register: *STUDENT*);
var enrollment: ABSTRACT;
course-name, quarter: *STRING*;
nbr-courses, year: *INTEGER*;
answer: *CHAR*;
begin **transaction begin**
nbr-courses := 0;
writeln(`Please enter the current year and quarter');
readln(year, quarter);
repeat
writeln(`Please enter the course name');
readln(course-name);
enroll(student-to-enroll, course-name, quarter, year);

```
            nbr-courses := nbr-courses + 1;
            writeln(`Register for another course (y/n) ?');
            readln(answer);
    until (answer = 'n' or nbr-courses > = 5) ;
    end;
    end;
```

Sometimes, it is of interest to the users of a database to retrieve or store global information about a category rather than specific information about its objects. Such information about categories can be manipulated by methods which are called **category methods**, and are a special case of object methods in that the objects upon which they are invoked are categories themselves.

Category methods are defined within the SBOODM by considering a special category *CATEGORY* whose objects are the categories of the schema. The category *CATEGORY* is an example of a special type of categories called **meta-categories**. Meta- categories are used together with meta-relations to manipulate the concepts of a schema. Formally, category methods are defined as follow:

Category method: This is an object method such that the category for which it is defined is the meta-category *CATEGORY*.

Example
The following is an example of a category method which counts the number of objects contained in a given category.

```
    function category-size(category-object: CATEGORY): INTEGER;
    var object: ABSTRACT;
        count: INTEGER;
    begin
    count := 0;
    for object in category-object do count := count + 1;
    category-size := count;
    end;
```

In this example, the method category-size had to repeatedly read the database in order to determine the number of objects that are members of some particular category. Obviously, *category-size* is an expensive operation in terms of number of times that the database must be accessed, especially if the size of the database is large. An alternative to this implementation is to have all global information that is of concern to the users of a category (e.g., the number of objects which are members of a category) associated with the category object itself via some relation.

3.6 Semantic Introduction to Databases

This sub-unit defines fundamental concepts of databases. These concepts are described here in terms of the **Semantic Binary Model** (SBM) of data. A data model is a convention for the specification of the logical structure of real-world information. The cornerstone of the contemporary theory and technology of databases was the development of the Relational Data Model. The recent development of the new generation of data models - the semantic models - offers a simple, natural, implementation-independent, flexible, and non-redundant specification of information. The word semantic means that this convention closely captures the meaning of user's information and provides a concise, high-level description of that information.

SBM is one of several existing semantic models. The various semantic models are roughly equivalent and have common principles, even though they somewhat differ in terminology and in the tools they use. SBM is simpler than most other semantic data models: it has a small set of

sufficient tools by which all of the semantic descriptors of the other models can be constructed. After mastering SBM, a systems analyst may wish to explore more complex semantic models.

Database:

This is an updatable storage of information of an application's world and managing software that conceals from the user the physical aspects of information storage and information representation. The information stored in a database is accessible at a logical level without involving the physical concepts of implementation.

Example
Neither a user nor a user program will try to seek the names of computer science instructors in track 13 of cylinder 5 of a disk or in "logical" record 225 of file XU17.NAMES.VERSION.12.84. Instead, the user will communicate with the database using some logical structure of the application's information.

Normally, a database should cover all the information of one application; there should not be two databases for one application.

Database management system, DBMS:

This is a general-purpose software system which can manage databases for a very large class of the possible application worlds.

Example
A DBMS is able to manage our university database and also completely different databases: an Internal Revenue Service database, an FBI WANTED database, a UN database on world geographical data, an Amtrak schedule, etc.

Instantaneous database:

This contains all the information represented in a database at a given instant. This includes the historic information which is still kept at that time. The actual information stored in the database changes from day to day. Most changes are additions of information to the database.

Example
A new student, a new instructor, new events of course offerings.
Fewer changes are deletions of information.
Historic information past the archival period; a course offering which was cancelled before it was given.

Some changes are replacements:

updates; correction of wrongly recorded information.

Example

Update of the address of a student; correction of the student's birth year (previously wrongly recorded).

Hence the life of a database can be seen as a sequence of instantaneous databases. The first one in the sequence is often the empty instantaneous database -- it is the state before any information has been entered.

Database model:

This is a convention of specifying the concepts of the real world in a form understandable by a DBMS. (Technically, it is an abstract data structure such that every possible instantaneous database of nearly every application's world can be logically represented by an instance of that data structure.)

The Semantic Binary Model is the most natural of the above models. It is the most convenient for specifying the logical structure of information and for defining the concepts of an application's world. In this text, the other models will be derived from the Semantic Binary Model. The Relational, Network and Hierarchical models are dominant in today's commercial market of database management systems.

Semantic Modelling

Categorization of objects:

Object: This is any item in the real world. It can be either a concrete object or an abstract object as follows.

Example
Consider the application world of a university.
I am an object, if I am of interest to the university. My name is an object. The Information Systems Department and its name "Information Systems Department" are two distinct objects.

Value or Concrete Object: This is a printable object, such as a number, a character string, or a date. A value can be roughly considered as representing itself in the computer or in any formal system.

Example
My name and the name "Computer Science Department" are concrete objects. The grade 70 which has been given to a student in a course is also a concrete object.

Abstract Object: This is a non-value object in the real world. An abstract object can be, for example, a tangible item (such as a person, a table, a country) or an event (such as an offering of a course by an instructor) or an idea (such as a course). Abstract objects cannot be represented directly in the computer. This term is also used for a user-transparent representation of such an object in the Semantic Binary Model.

Example
The Management Science Department, the student of the department whose name is Alex Johnson, and the course named "Chemistry" are three abstract objects.

Category: This is any concept of the application's real world which is a unary property of objects. At every moment in time such a concept is descriptive of a set of objects which possess the property at that time. Unlike the mathematical notion of a set, the category itself does not depend on its objects: the objects come and go while the meaning of the category is preserved in time. Conversely, a set does depend on its members: the meaning of set changes with the ebb and flow of its members.

Categories are usually named by singular nouns.

Example
STUDENT is a category of abstract objects. The set of all the students relevant to the application today is different from such a set tomorrow, since new students will arrive or will become relevant. However, the concept STUDENT will remain unaltered.

An object may belong to several categories at the same time.

Example
One object may be known as a person and at the same time as an instructor and as a student.
Example
Some of the categories in the world of our university are: INSTRUCTOR, PERSON, COURSE, STUDENT, DEPARTMENT.

Disjoint Categories: Two categories are disjoint if no object may simultaneously be a member of both categories. This means that at every point in time the sets of objects corresponding to two disjoint categories have an empty intersection.

Example
The categories STUDENT and COURSE are disjoint; so are COURSE and DEPARTMENT (even though there may be two different objects, a course and a department, both named "Physics"). The categories INSTRUCTOR and STUDENT are not disjoint (Figure 1-1); neither are INSTRUCTOR and PERSON.

Subcategory: A category is a subcategory of another category if at every point in time every object of the former category should also belong to the latter. This means that at every point in time the set of objects corresponding to a category contains the set of objects corresponding to any subcategory of the category.

Example
The category STUDENT is a subcategory of the category PERSON. The category INSTRUCTOR is another subcategory of the category PERSON (Figure 1-2).

Abstract category: This is a category whose objects are always abstract.

Concrete category or category of values: This is a category whose objects are always concrete.

Example
STUDENT and COURSE are abstract categories. STRING, NUMBER, and DIGIT are concrete categories.

Many concrete categories, such as NUMBER, STRING, and BOOLEAN, have constant-in-time sets of objects. Thus, those concrete categories are actually indistinguishable from the corresponding sets of all numbers, all strings, and the Boolean values ({TRUE, FALSE}).

Finite category: This is a category is finite if at no point in time an infinite set of objects may correspond to it in the application's world.

Example
The categories STUDENT, COURSE, and DIGIT are finite. The category NUMBER may be infinite. *139q Every abstract category is finite.*

Example
The database has a finite size. We cannot have abstract category POINT containing information about every point in a plane.

Binary relations

Binary relation: This is any concept of the application's real world which is a binary property of objects, that is, the meaning of a relationship or connection between two objects.

Example
WORKS-IN is a relation relating instructors to departments. MAJOR-DEPARTMENT relates students to departments. NAME is a relation relating persons to strings. BIRTH-YEAR is a relation relating persons to numbers.
At every moment in time, the relation is descriptive of a set of pairs of objects which are related at that time. The meaning of the relation remains unaltered in time, while the sets of pairs of objects corresponding to the relation may differ from time to time, when some pairs of objects cease or begin to be connected by the relation.

Notation: xRy means that object x is related by the relation R to object y.

Example
To indicate that an instructor i works in a department d, we write:

i WORKS-IN d

Types of binary relations: m:m, m:1, 1:m, 1:1.

1. A binary relation *R* is many-to-one (*m:1*, functional) if at no point in time *xRy* and *xRz* where *y=z*.

Example
BIRTH-YEAR is an *m:1* relation because every person has only one year of birth:

$person_1$	BIRTH-YEAR	1970
$person_2$	BIRTH-YEAR	1970
$person_3$	BIRTH-YEAR	1969
$person_4$	BIRTH-YEAR	1965

Table 2

Example
MAJOR-DEPARTMENT is also an *m:1* relation, since every student has at most one major department, as in above example.

2. A binary relation *R* is one-to-many (*1:m*) if at no point in time *xRy* and *zRy* where *x=z*.

Example
The relation MAJOR-DEPARTMENT is not 1:m, since a department may have many major students.
If, instead of the relation MAJOR-DEPARTMENT, we have the relation MAJOR-STUDENT between departments and students, then this relation would be 1:m, since every student can have at most one major department.

3. Relations which are of neither of the above types are called proper many-to-many (*m:m*).

Example
WORKS-IN is a proper m:m relation because every instructor can work in many departments and every department may employ many instructors, as in Figure 1-4.

4. A binary relation which is both *m:1* and *1:m* (always) is called one-to-one (*1:1*).

Example
If courses are identified by their names, then the relation COURSE-NAME is 1:1, meaning that every course has at most one name, and no character string is the name of two different courses, as in Figure 1-5.

Example
Suppose that in the current situation in our real world, the following is true:
Every registered person has at most one name, and no two persons have the same name.
This does not mean that NAME is a 1:1 relation between persons and strings. NAME would be a 1:1 relation if condition 'a' were true at all times: past, present, and future.

5. A binary relation is proper *m:1* if it is *m:1* and not *1:1*.

6. A binary relation is proper *1:m* if it is *1:m* and not *1:1*.

Example
All of the types of relations mentioned in the previous example are proper.
Since the COURSE-NAME is 1:1, it is also 1:m, m:1, and m:m. Since this relation is proper 1:1, it cannot be proper 1:m, proper m:1, or proper m:m.

Categories as domains and ranges of relations

Domain and range of a binary relation:

Domain of relation R: A category C that satisfies the following two conditions: **a.** Whenever xRy, then x belongs to C (at every point in time for every pair of objects) **b.** No proper subcategory of C satisfies condition a Range of relation R - a category C that satisfies: **a.** Whenever xRy, then y belongs to C (at every point in time for every pair of objects) **b.** No proper subcategory of C satisfies a

Example
The domain of COURSE-NAME is the category COURSE and its range is the category STRING. The domain of WORKS-IN is INSTRUCTOR and the range is DEPARTMENT.

Total binary relation: A relation *R* whose domain is *C* is total if at all times for every object *x* in *C* there exists an object *y* such that *xRy*. (At different times different objects *y* may be related to a given object *x*.)

Note: No relation needs to be total on its domain.

Example
Although the domain of the relation BIRTH-DATE is the category PERSON, the date of birth of some relevant persons is irrelevant or unknown. Thus, the relation BIRTH-DATE is not total.

Attributes: Some binary relations are often called attributes. This is a functional relation (i.e., m:1 or 1:1) whose range is a concrete category.

Example

- *[] first-name - attribute of PERSON, range: String (m:1)*
- *[] birth-year - attribute of PERSON, range: 1880..1991 (m:1) The phrase ``a is an attribute of C'' means: a is an attribute, and its domain is the category C.*

Example
Last-name, first-name, and birth-year are attributes of PERSON.

Non-binary relationships: These are real-world relationships that bind more than two objects in different roles.

Example
There is a relationship between an instructor, a course, and a quarter in which the instructor offers the course. Such complex relationships are regarded in the Semantic Binary Model as groups of several simple relationships.
Example
The non-binary relationship of the previous example is represented in the Semantic Binary Model by a fourth object, an offering, and three binary relations between the offering and the instructor, the quarter, and the course.

In general, the Semantic Binary Model represents any non-binary relation as:

a. An abstract category of events. Each event symbolizes the existence of a relationship between a group of objects.

b. Functional binary relations, whose domain is category (a).

Each of those functional binary relations corresponds to a

role played by some objects in the non-binary relation.

Thus, the fact that objects $x_1, ..., x_n$ participate in an n-ary relation R in roles $R_1, ..., R_n$ is represented by:

a. An object e in the category R'

b. Binary relationships $eR_1 x_1, ..., eR_n x_n$

Example
The information about a course offered by an instructor during a quarter could be considered a ternary relation between instructors, courses, and quarters. In the Semantic Binary Model, we solve this problem by representing this information as a category COURSE-OFFERING and three functional relations from COURSE-OFFERING: THE-INSTRUCTOR, THE-COURSE, and THE-QUARTER.

Instructor i has offered course c in quarter q if and only if there exists a course-offering o, such that:

- o THE-INSTRUCTOR i
- o THE-COURSE c
- o THE-QUARTER q

4.0 Conclusion

The exploration of this unit has clearly revealed to individuals and organisations, how stored symbols relate to the real world, that is, how models depict the true representation of the real world.

5.0 Summary

In this unit we have learnt that:

- ❖ A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.
- ❖ The applications of semantic data models include planning of data resources, building of shareable databases, evaluation of vendor software, and integration of existing databases.
- ❖ Semantic introduction to databases defines fundamental concepts of databases, which are described in terms of semantic binary model.

6.0 Tutor Marked Assignment

1. (a) What do you understand by the term semantic data models
   (b) Mention and explain the applications of semantic data models
2. Write explicitly on semantic introduction to databases.

7.0 Further Reading and Other Resources

Jack, Colin, and Colin Keillor. "Semantic Data Models." Unpublished essay. School of Computing, Napier University. 15 Oct. 2006 <http://www.soc.napier.ac.uk/module.php3

Semantic Data Model. 13 Aug. 2001. COMTECO Ltd. 15 Oct. 2006 <http://www.comteco.ru/EN/DATAMODE/datamode.htm

http://en.wikipedia.org/wiki/Category:Wikipedia_articles_incorporating_text_from_the_National_Institute_of_Standards_and_Technology

Y. Tina Lee (1999). "Information modelling from design to implementation" National Institute of Standards and Technology.

Module 2 Semantic Data Modelling

Unit 3 Semantic Data Modelling Concepts

1.0 Introduction

Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the instances, it is required that the meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardise such relation types. This means that the second kind of semantic data models enable that the instances express facts that include their own meaning.

The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies that in general they have a wider applicability than relational or object oriented databases.

2.0 Objective

At the end of this unit, you should be able to:

- ❖ Explain the essential element of semantic data language.
- ❖ Mention different organisations of semantic models
- ❖ Describe Semantic Data in the Database
- ❖ Explain Semantic Annotation, Indexing, and Retrieval
- ❖ Concepts of entities, design and indexing in semantic modelling

3.0 Overview of Semantic Data language

A new and efficient data language (more suitable for end users than SQL) respecting data model structure and avoiding the pitfalls (joins and NULLs) of SQL, was developed by **Johan ter Bekke**. Here is an overview of the main data query and manipulation commands:

Command	**Description**
Get	to retrieve data
Extend	to derive/compute intermediate data
Value	to store one (possibly derived) value
Insert	to add a record
Update	to change data
Cascade	to process data in a recursive way
Delete	to remove data

Relational join pitfall

Before discussing semantic data language principles it is essential to discuss a main weakness of the relational language SQL leading to a pitfall, especially for non-expert users. In particular we discuss the problems generated by the relational join operation when combined with GROUP BY. In order to be able to discuss that pitfall we use the following simple data model:

*employee (**emp#**, name, address, town, birth_date, function, dept#);*
*department (**dept#**, address, town, main_task);*

The semantic equivalent of this model:

type employee = name, address, town, birth_date, function, department.
type department = address, town, main_task.

Figure 24

We suppose that this data model is used by an organization located in Abuja wanting to determine which departments (dept#) have the smallest number of employees living in Abuja. This information can be relevant when departments offer a compensation for the travelling costs of their employees. A user could think that this problem is solved by the following SQL-query:

```
CREATE VIEW temporal (dept#, number) AS
SELECT department.dept#, COUNT (emp#)
FROM employee, department
WHERE employee.dept# = department.dept#
```

```
AND employee.town = Abuja
GROUP BY department.dept#;
SELECT dept#
FROM temporal
WHERE number IN (SELECT MIN (number) FROM temporal);
```

If a department does not have any employee living in Abuja, its dept# will not be present in the temporal table created by the view temporal. A more extended query solves this problem, using a union of two sub sets:

```
 CREATE VIEW temporal (dept#, number) AS
 (SELECT department.dept#, COUNT (emp#)
 FROM employee, department
 WHERE employee.dept# = department.dept#
 AND employee.town = Abuja
 GROUP BY department.dept#)
UNION(SELECT dept#, 0 FROM department WHERE dept#
NOT IN (SELECT dept# FROM employee WHERE town =
Abuja));
SELECT dept# FROM temporal WHERE number IN (SELECT
MIN (number) FROM temporal);
```

The last solution guarantees that, if a department does not have any employee living in Abuja the value of the derived attribute temporal number is 0 for such a department. Now the derived information will be correct (and complete) irrespective the contents of the database.

The semantic language does not allow for join-operations; it requires applying paths really existing in the underlying data model. The first step is to derive the required number of employees per department, using the temporal attribute department its number:

extend department *with* number = *count* employee
where town = Abuja *per* department.

value minimum = *min* department *its* number.

get department *where* number = minimum.

The term per means the same as for each, which most probably is easier to understand then GROUP BY: if an empty sub set of data is involved in a join combined with the GROUP BY construct then an incomplete, thus incorrect query result is possible! Here we have the strange situation that a seemingly semantically correct SQL-query produces a result of which the correctness depends on the actual contents of the database!

In the semantic approach NULL-values do not occur in the database. What would be the query result of counting Amsterdam-employees per department if we allow that the value of the attribute employee its department can be NULL? Moreover, allowing NULL-values would lead to many problems because in many queries we have to walk through data paths existing in the underlying data model. The above mentioned extend operation applies a path consisting of only one attribute: employee its department. However, longer paths may be used as in the following example where the path employee its department its town is involved in a query about commuters:

get employee *its* name, address, department, town, department *its* town
where not town = department *its* town.

In many cases users want to retrieve information about data sets, but it is also possible and simple to get information about some specific instance, for example an employee with the identification 233:

get employee 233 *its* name, address, town, department.

3.1 Organization of Semantic Data Models.

The piece of the "real-world" represented by a semantic model is referred to as an enterprise. A semantic schema represents important elements within an enterprise and shows the structural interrelationships among those elements. Virtually all semantic data models offer a diagrammatic construct for conceptualizing schemas. The fundamental components used by semantic models to structure data are objects, atomic and constructed types, attributes, ISA relationships, and derived schema components. We begin by discussing objects, atomic types, constructed types and so on.

(1) Objects − Elements within an enterprise are represented by objects or entities. It is generally assumed that these objects can be any unstructured data such as strings, integers, and reals, or, in the case of multimedia enterprises, text, voice and image data.

(2) Atomic types − The direct representation of object types, distinct from their attributes, is essential to semantic modelling. As the name implies, atomic types correspond to classes of simple, non-aggregate objects in an enterprise. The Hypertext Community schema in Fig. 1 shows the organization of object types in a hypothetical enterprise. In it, two atomic types, PERSON and EMPLOYER, are represented. These are depicted using triangles to indicate that they are abstract or nonprintable types.

Abstract types generally correspond directly to physical or conceptual

objects in an enterprise. Depending upon a particular model's implementation, these types may not be visible to the user. In contrast, Printable types, such as PNAME and OCCUPATION, would have an external representation accessible to the user. The active domain of an atomic type holds all objects of that type currently in the database.

(3) Constructed types − Perhaps the most important feature of semantic models is their ability to construct complex object types from atomic types. Aggregation and grouping, also called association, are the most common type constructors in the semantic literature.

An aggregation is a composite type formed from other types already existing in the schema. Mathematically, an aggregation is an ordered n-tuple, and, in an instance of the schema the active domain of an aggregation type will be a subset of the Cartesian product of the active domains assigned to its underlying nodes.

The aggregation abstraction allows users to focus on a complex data type while ignoring its component parts. The grouping constructor, on the other hand, is used to represent sets of objects of the same type.

Mathematically, a grouping is a finitary power set. In an instance, the active domain of a grouping type will consist of a set of objects, each of which is a finite subset of the active domain of the underlying node.

Although aggregation and grouping define new object types from previously defined types, they are fundamentally distinct abstractions. Aggregation, in effect, provides a means for specifying the attributes of a

new object type (see below), whereas grouping defines a type whose value will be a set of objects of a particular type. In most semantic models supporting aggregation and grouping, there can be no directed or undirected cycles of type constructor edges.

(4) Attributes−Another fundamental aspect of semantic models is their ability to represent interrelational dependencies or connections among object types. These properties are called attributes or relationships. Attributes are generally specified explicitly by the user to correspond to facts about an enterprise. In the Hypertext Community schema, PERSON has three attributes: HAS-NAME, LIVES-AT, and USES.

In this schema, attributes are represented by arrows that originate at the domain of each attribute and terminate at its range. That is to say, the HAS-NAME attribute maps an object of type PERSON to a printable object of type PNAME with similar mappings implied by the other attributes. A range object can be referred to as the value of the mapping attribute.
In a more formal sense, there are several ways attributes can be expressed in semantic models. The HAS-NAME attribute is an example of a one-argument attribute, i.e. it is a directed, binary relationship between two types. Some models allow n-argument attributes, which define a directed relationship between a set of n types and one type.
Attributes can also be either single-valued or multi-valued. An ADDRESS, for example, can be the residence of several PERSONs. In our schema, single- and multi-valued relationships are distinguished by one- and two-

headed arrows respectively. In an instance of the schema, a mapping is assigned to each attribute. The mapping will consist of
either a binary or (n+1)-ary relation, depending on the number of arguments n in the attribute. The domain of the mapping is the Cartesian product of the active domain(s) of the attribute's source(s), and the range is the active domain of the attribute's target. In the case of single-valued attributes, such as HAS-NAME, the mapping must be a function in the strict mathematical sense.

The distinctions between various forms of type constructors and attributes help explain some of the variability one sees in the expressiveness of semantic models. For example, there is a close correspondence between the semantics of a multi-valued attribute and the semantics of a single-valued attribute whose range is a constructed grouping type. Likewise, a one-argument attribute whose domain is an aggregation and an n-argument attribute are similar.

It should be clear that several ways of representing essentially the same data interrelationships can exist given a full range of modelling functionality. Most existing models, however, only support multi-valued attributes and do not permit an attribute to map to a grouping type. What may not be quite so obvious is that the semantics of object identity are strongly influenced by the abstractions chosen in a representation.

There is a final point about attributes. Some semantic models draw a fine distinction between attributes and relationships. In these cases, attributes are considered relationships in which the values are atomic. In addition, some models allow relationships, but not attributes, to have attributes of their

own, just like other objects. In the remainder of this paper, we use the two terms interchangeably.

(5) ISA relationships − Virtually all semantic models have the ability to represent ISA or supertype/subtype relationships. Generally speaking, an ISA relationship from a subtype to a supertype indicates that each object associated with the subtype is also associated with the supertype.
In most semantic models, subtypes inherit their supertype's attributes and can have attributes that are not shared by their supertype. Since semantic models usually allow undirected or weak cycles to exist among these relationships, they form what is often referred to as
the schema's ISA network.

ISA relationships are used in semantic models for two closely related purposes. First, they can represent one or more overlapping subtypes of a type. Second, they can be used to form types that contain a union of disjoint types already present in the schema. Historically, semantic models have used a single kind of ISA relationship for both purposes.
Recent research, however, has differentiated several kinds of ISA relationships, such as subset and generalization that allow the subtle semantics of set containment and partitioning to be expressed.

(6) Derived schema components − Derived schema components, also called derived data, are a basic mechanism for data abstraction and encapsulation in many semantic models. Derivation allows information to be incorporated into a database schema that is itself computed from other

information in the schema. Derived schema components consist of a structural specification for holding the derived information and a derivation rule describing how the structure is to be filled. The structure, in most cases, is either a derived subtype or a derived attribute.

Generally speaking, derived data are automatically updated as required by updates to other parts of the schema. Objects, atomic and constructed types, attributes, ISA relationships, and derived schema components are the fundamental abstractions offered by the semantic modelling paradigm. It is important to note that much of the expressive power of semantic models comes from the fact that these constructs can usually be used recursively.

From a broader perspective, though, the effectiveness of semantic modelling is influenced by several other important factors. These include such things as combining restrictions, static integrity constraints, and the manipulation languages that enable users to interact with data.

(1) Combining restrictions − Most semantic models do not allow arbitrary combinations of the basic constructs. The most prominent of these restrictions affect the combining of ISA relationships. For example, directed cycles of ISA edges really make no sense and are usually not permitted. Generally speaking, combining restrictions assure that schemas are not redundant or ambiguous and capture the natural intuition about objects.

(2) Static integrity constraints − In general, semantic models express in a structural manner the most important types of relational integrity

constraints. There are, however, a variety of relationships and properties of relationships that cannot be directly represented with the abstractions presented thus far. If objects are connected through relationships, then insertion, deletion, or modification of one object can still impact others. For this reason, many semantic models provide mechanisms for specifying integrity constraints, which in essence, assure that data associated with different parts of a schema are consistent according to some criteria. We will return to this issue in our discussion section.

(3) Manipulation languages − The data structuring capabilities discussed so far would normally be supported by a data definition language associated with a specific model. In addition, the model would be expected to have a corresponding data manipulation language that allows users to interact with the database. There are three fundamental capabilities that differentiate a semantic data manipulation language from a manipulation language for a traditional, record-oriented model: its ability to query abstract types, manipulate attributes, and manage derived data.

3.1.1 Semantic Data in the Database

There is one universe for all semantic data stored in the database. All triples are parsed and stored in the system as entries in tables under the MDSYS schema. A triple {subject, property, and object} is treated as one database object. As a result, a single document containing multiple triples results in multiple database objects.

All the subjects and objects of triples are mapped to nodes in a semantic data network, and properties are mapped to network links that have their start node and end node as subject and object, respectively. The possible node types are blank nodes, URIs, plain literals, and typed literals.

The following requirements apply to the specifications of URIs and the storage of semantic data in the database:

- A subject must be a URI or a blank node.
- A property must be a URI.
- An object can be any type, such as a URI, a blank node, or a literal. (However, null values and null strings are not supported.)

3.1.2 Metadata for Models

The MDSYS.SEM_MODELS view contains information about all models defined in the database. When you create a model using the SEM_APIS.CREATE_SEM_MODEL procedure, you specify a name for the model, as well as a table and column to hold references to the semantic data, and the system automatically generates a model ID.

Oracle maintains the MDSYS.SEM_MODELS view automatically when you create and drop models. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view.

The MDSYS.SEM_MODELS view contains the columns shown in Table 3

Table 3. MDSYS.SEM_MODELS View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the model.
MODEL_ID	NUMBER	Unique model ID number, automatically generated.
MODEL_NAME	VARCHAR2(25)	Name of the model.
TABLE_NAME	VARCHAR2(30)	Name of the table to hold references to semantic data for the model.
COLUMN_NAME	VARCHAR2(30)	Name of the column of type SDO_RDF_TRIPLE_S in the table to hold references to semantic data for the model.
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Name of the tablespace to be used for storing the triples for this model.

When you create a model, a view for the triples associated with the model is also created under the MDSYS schema. This view has a name in the format RDFM_model-name, and it is visible only to the owner of the model and to users with suitable privileges. Each MDSYS.SEMM_model-name view contains a row for each triple (stored as a link in a network), and it has the columns shown in Table 4.

Table 4 MDSYS.SEMM_model-name View Columns

Column Name	Data Type	Description
P_VALUE_ID	NUMBER	The VALUE_ID for the text value of the predicate of the triple. Part of the primary key.

Column Name	Data Type	Description
START_NODE_ID	NUMBER	The VALUE_ID for the text value of the subject of the triple. Also part of the primary key.
CANON_END_NODE_ID	NUMBER	The VALUE_ID for the text value of the canonical form of the object of the triple. Also part of the primary key.
END_NODE_ID	NUMBER	The VALUE_ID for the text value of the object of the triple
MODEL_ID	NUMBER	The ID for the RDF graph to which the triple belongs. It logically partitions the table by RDF graphs.
COST	NUMBER	(Reserved for future use)
CTXT1	NUMBER	(Reserved for future use)
CTXT2	VARCHAR2(4000)	(Reserved for future use)
DISTANCE	NUMBER	(Reserved for future use)
EXPLAIN	VARCHAR2(4000)	(Reserved for future use)
PATH	VARCHAR2(4000)	(Reserved for future use)
LINK_ID	VARCHAR2(71)	Unique triple identifier value. (It is currently a computed column, and its definition may change in a future release.)

Note:

In Table 4, for columns P_VALUE_ID, START_NODE_ID, END_NODE_ID, and CANON_END_NODE_ID, the actual ID values are computed from the corresponding lexical values. However, a lexical value may not always map to the same ID value.

3.1.3 Statements

The MDSYS.RDF_VALUE$ table contains information about the subjects, properties, and objects used to represent RDF (Resource Description Framework) statements. It uniquely stores the text values (URIs or literals) for these three pieces of information, using a separate row for each part of each triple. Oracle maintains the MDSYS.RDF_VALUE$ table automatically. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view.

The RDF_VALUE$ table contains the columns shown in Table 5.

Table 5 MDSYS.RDF_VALUE$ Table Columns

Column Name	Data Type	Description
VALUE_ID	NUMBER	Unique value ID number, automatically generated.
VALUE_TYPE	VARCHAR2(10)	The type of text information stored in the VALUE_NAME column. Possible values: UR for URI, BN for blank node, PL for plain literal, PL@ for plain literal with a language tag, PLL for plain long literal, PLL@ for plain long literal with a language tag, TL for typed literal, or TLL for typed long literal. A long literal is a literal with more than 4000 bytes.
VNAME_PREFIX	VARCHAR2(4000)	If the length of the lexical value is 4000 bytes or less, this column stores a prefix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_PREFIX function can be used for prefix computation. For example, the prefix for the portion of the lexical value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> without the angle brackets is http://www.w3.org/1999/02/22-rdf-syntax-ns#.
VNAME_SUFFIX	VARCHAR2(512)	If the length of the lexical value is 4000 bytes or less, this column stores a suffix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_SUFFIX function

Column Name	Data Type	Description
		can be used for suffix computation. For the lexical value mentioned in the description of the VNAME_PREFIX column, the suffix is type.
LITERAL_TYPE	VARCHAR2(4000)	For typed literals, the type information; otherwise, null. For example, for a row representing a creation date of 1999-08-16, the VALUE_TYPE column can contain TL, and the LITERAL_TYPE column can contain http://www.w3.org/2001/XMLSchema#date.
LANGUAGE_TYPE	VARCHAR2(80)	Language tag (for example, fr for French) for a literal with a language tag (that is, if VALUE_TYPE is PL@ or PLL@). Otherwise, this column has a null value.
CANON_ID	NUMBER	The ID for the canonical lexical value for the current lexical value. (The use of this column may change in a future release.)
COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the lexical value. (The use of this column may change in a future release.)
CANON_COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the canonical lexical value. (The use of this column may change in a future release.)
LONG_VALUE	CLOB	The character string if the length of the lexical value is greater than 4000 bytes. Otherwise, this column has a null value.
VALUE_NAME	VARCHAR2(4000)	This is a computed column. If length of the lexical value is 4000 bytes or less, the value of this column is the concatenation of the values of VNAME_PREFIX column and the VNAME_SUFFIX column.

3.1.4 Triple Uniqueness and Data Types for Literals

Duplicate triples are not stored in the database. To check if a triple is a duplicate of an existing triple, the subject, property, and object of the incoming triple are checked against triple values in the specified model. If the incoming subject, property, and object are all URIs, an exact match of

their values determines a duplicate. However, if the object of incoming triple is a literal, an exact match of the subject and property, and a value (canonical) match of the object, determine a duplicate. For example, the following two triples are duplicates:

<eg:a> <eg:b> "123"^^http://www.w3.org/2001/XMLSchema#int
<eg:a> <eg:b> "123"^^http://www.w3.org/2001/XMLSchema#unsignedByte

The second triple is treated as a duplicate of the first, because

"123"^^http://www.w3.org/2001/XMLSchema#int has an equivalent value (is canonically equivalent) to "123"^^http://www.w3.org/2001/XMLSchema#unsignedByte. Two entities are canonically equivalent if they can be reduced to the same value.

To use a non-RDF example, `A*(B-C)`, `A*B-C*A`, `(B-C)*A`, and `-A*C+A*B` all convert into the same canonical form.

Value-based matching of lexical forms is supported for the following data types:

- STRING: plain literal, xsd:string and some of its XML Schema subtypes
- NUMERIC: xsd:decimal and its XML Schema subtypes, xsd:float, and xsd:double. (Support is not provided for float/double INF, -INF, and NaN values.)
- DATETIME: xsd:datetime, with support for time zone. (Without time zone there are still multiple representations for a single value, for example, "2004-02-18T15:12:54" and "2004-02-18T15:12:54.0000".)
- DATE: xsd:date, with or without time zone
- OTHER: Everything else. (No attempt is made to match different representations).

Canonicalization is performed when the time zone is present for literals of type xsd:time and xsd:dateTime.

The following namespace definition is used: `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

The first occurrence of a literal in the RDF_VALUES table is taken as the canonical form and given the VALUE_TYPE value of CPL, CPL@, CTL, CPLL, CPLL@, or CTLL as appropriate; that is, a C for canonical is prefixed to the actual value type. If a literal with the same canonical form (but a different lexical representation) as a previously inserted literal is inserted into the RDF_VALUES table, the VALUE_TYPE value assigned to the new insert is PL, PL@, TL, PLL, PLL@, or TLL as appropriate.

Canonically equivalent text values having different lexical representations are thus stored in the RDF_VALUES table; however, canonically equivalent triples are not stored in the database.

3.1.5 Subjects and Objects

RDF subjects and objects are mapped to nodes in a semantic data network. Subject nodes are the start nodes of links, and object nodes are the end nodes of links. Non-literal nodes (that is, URIs and blank nodes) can be used as both subject and object nodes. Literals can be used only as object nodes.

3.1.6 Blank Nodes

Blank nodes can be used as subject and object nodes in the semantic network. Blank node identifiers are different from URIs in that they are scoped within a semantic model. Thus, although multiple occurrences of the same blank node identifier within a single semantic model necessarily refer to the same resource, occurrences of the same blank node identifier in two different semantic models do not refer to the same resource.

In an Oracle semantic network, this behaviour is modelled by requiring that blank nodes are always reused (that is, are used to represent the same resource if the same blank node identifier is used) within a semantic model,

and never reused between two different models. Thus, when inserting triples involving blank nodes into a model, you must use the SDO (Service Data Objects)_RDF_TRIPLE_S constructor that supports reuse of blank nodes.

3.1.7 Properties

Properties are mapped to links that have their start node and end node as subjects and objects, respectively. Therefore, a link represents a complete triple. When a triple is inserted into a model, the subject, property, and object text values are checked to see if they already exist in the database. If they already exist (due to previous statements in other models), no new entries are made; if they do not exist, three new rows are inserted into the RDF_VALUES table (described in Section 5).

3.1.8 Inferencing: Rules and Rulebases

Inferencing is the ability to make logical deductions based on rules. Inferencing enables you to construct queries that perform semantic matching based on meaningful relationships among pieces of data, as opposed to just syntactic matching based on string or other values. Inferencing involves the use of rules, either supplied by Oracle or user-defined, placed in rulebases.

Figure 25 shows triple sets being inferred from model data and the application of rules in one or more rulebases. In this illustration, the database can have any number of semantic models, rulebases, and inferred

triple sets, and an inferred triple set can be derived using rules in one or more rulebases.

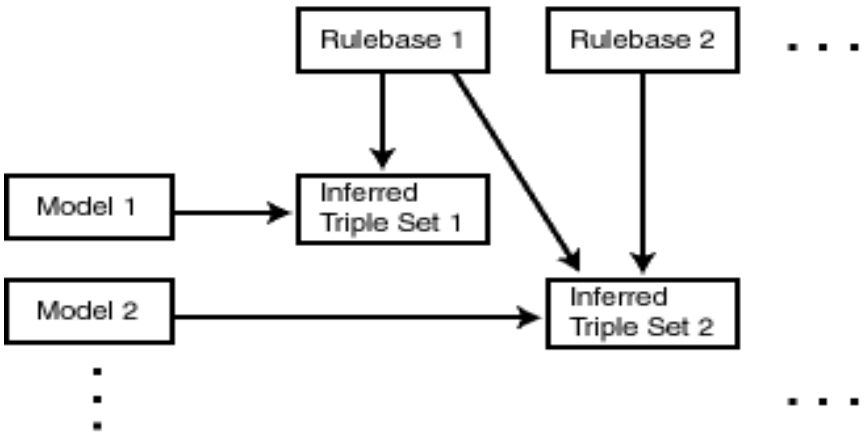

Figure 25 Inferencing

Description of "Figure 25 Inferencing"

A rule is an object that can be applied to draw inferences from semantic data. A rule is identified by a name and consists of:

- An IF side pattern for the antecedents
- An optional filter condition that further restricts the subgraphs matched by the IF side pattern
- A THEN side pattern for the consequents

For example, the rule that a chairperson of a conference is also a reviewer of the conference could be represented as follows:

```
('chairpersonRule', -- rule name
'(?r :ChairPersonOf ?c)', -- IF side pattern
 NULL,  -- filter condition
'(?r :ReviewerOf ?c)', -- THEN side pattern
 SEM_ALIASES (SEM_ALIAS('', 'http://some.org/test/')))
```

In this case, the rule does not have a filter condition, so that component of the representation is NULL. Note that a THEN side pattern with more than one triple can be used to infer multiple triples for each IF side match.

A rulebase is an object that contains rules. The following Oracle-supplied rulebases are provided:

- RDFS
- RDF (a subset of RDFS)
- OWLSIF (empty)
- RDFS++ (empty)
- OWLPRIME (empty)

The RDFS and RDF rulebases are created when you call the SEM_APIS.CREATE_SEM_NETWORK procedure to add RDF support to the database. The RDFS rulebase implements the RDFS entailment rules, as described in the World Wide Web Consortium (W3C) RDF Semantics document at http://www.w3.org/TR/rdf-mt/. The RDF rulebase represents the RDF entailment rules, which are a subset of the RDFS entailment rules. You can see the contents of these rulebases by examining the MDSYS.SEMR_RDFS and MDSYS.SEMR_RDF views.

You can also create user-defined rulebases using the SEM_APIS.CREATE_RULEBASE procedure. User-defined rulebases enable you to provide additional specialized inferencing capabilities.

For each rulebase, a system table is created to hold rules in the rulebase, along with a system view with a name in the format MDSYS.SEMR_rulebase-name (for example, MDSYS.SEMR_FAMILY_RB for a rulebase named FAMILY_RB). You

must use this view to insert, delete, and modify rules in the rulebase. Each MDSYS.SEMR_rulebase-name view has the columns shown in Table 6 below.

Table 6 MDSYS.SEMR_rulebase-name View Columns

Column Name	Data Type	Description
RULE_NAME	VARCHAR2(30)	Name of the rule
ANTECEDENTS	VARCHAR2(4000)	IF side pattern for the antecedents
FILTER	VARCHAR2(4000)	Filter condition that further restricts the subgraphs matched by the IF side pattern. Null indicates no filter condition is to be applied.
CONSEQUENTS	VARCHAR2(4000)	THEN side pattern for the consequents
ALIASES	SEM_ALIASES	One or more namespaces to be used. (The SEM_ALIASES data type is described in Section 1.6.)

Information about all rulebases is maintained in the MDSYS.SEM_RULEBASE_INFO view, which has the columns shown in Table 7 and one row for each rulebase.

Table 7. MDSYS.SEM_RULEBASE_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rulebase
RULEBASE_NAME	VARCHAR2(25)	Name of the rulebase

Column Name	Data Type	Description
RULEBASE_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the rulebase
STATUS	VARCHAR2(30)	Contains VALID if the rulebase is valid, INPROGRESS if the rulebase is being created, or FAILED if a system failure occurred during the creation of the rulebase.

Example 1-1

creates a rulebase named family_rb, and then inserts a rule named grandparent_rule into the family_rb rulebase. This rule says that if a person is the parent of a child who is the parent of a child, that person is a grandparent of (that is, has the grandParentOf relationship with respect to) his or her child's child. It also specifies a namespace to be used.

Example 1-1 Inserting a Rule into a Rulebase

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');

NSERT INTO mdsys.semr_family_rb VALUES('grandparent_rule',
'(?x :parentOf ?y) (?y :parentOf ?z)',
 NULL,
 '(?x :grandParentOf ?z)',
SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')));
```

You can specify one or more rulebases when calling the SEM_MATCH table function (described in Section 1.6), to control the behavior of queries against semantic data. Example 1-2 refers to the family_rb rulebase and to the grandParentOf relationship created in Example 1-1, to find all grandfathers (grandparents who are male) and their grandchildren.

Example 1-2 Using Rulebases for Inferencing

```
-- Select all grandfathers and their grandchildren from the family model.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x, y
  FROM TABLE(SEM_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null));
```

3.1.9 Rules Indexes

A rules index is an object containing precomputed triples that can be inferred from applying a specified set of rulebases to a specified set of models. If a SEM_MATCH query refers to any rulebases, a rules index must exist for each rulebase-model combination in the query.

To create a rules index, use the SEM_APIS.CREATE_RULES_INDEX procedure. To drop (delete) a rules index, use the SEM_APIS.DROP_RULES_INDEX procedure.

When you create a rules index, a view for the triples associated with the rules index is also created under the MDSYS schema. This view has a name in the format SEMI_rules-index-name, and it is visible only to the owner of the rules index and to users with suitable privileges. Each MDSYS.SEMI_rules-index-name view contains a row for each triple (stored as a link in a network), and it has the same columns as the SEMM_model-name view, which is described in Table 4 above.

Information about all rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_INFO view, which has the columns shown in Table 8 and one row for each rules index.

Table 8 MDSYS.SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rules index
INDEX_NAME	VARCHAR2(25)	Name of the rules index
INDEX_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the rules index
STATUS	VARCHAR2(30)	Contains VALID if the rules index is valid, INVALID if the rules index is not valid, INCOMPLETE if the rules index is incomplete (similar to INVALID but requiring less time to re-create), INPROGRESS if the rules index is being created, or FAILED if a system failure occurred during the creation of the rules index.
MODEL_COUNT	NUMBER	Number of models included in the rules index
RULEBASE_COUNT	NUMBER	Number of rulebases included in the rules index

Information about all database objects, such as models and rulebases, related to rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_DATASETS view. This view has the columns shown in Table 9 and one row for each unique combination of values of all the columns.

Table 9 MDSYS.SEM_RULES_INDEX_DATASETS View Columns

Column Name	Data Type	Description
INDEX_NAME	VARCHAR2(25)	Name of the rules index
DATA_TYPE	VARCHAR2(8)	Type of data included in the rules index. Examples: `MODEL` and `RULEBASE`

Column Name	Data Type	Description
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

Example 1-3
creates a rules index named `family_rb_rix_family`, using the `family` model and the `RDFS` and `family_rb` rulebases.

Example 1-3 Creating a Rules Index

```
BEGIN
 SEM_APIS.CREATE_RULES_INDEX(
'rdfs_rix_family',
SEM_Models('family'),
SEM_Rulebases('RDFS','family_rb'));
END;
/
```

3.1.10 Virtual Models

A virtual model is a logical graph that can be used in a SEM_MATCH query. A virtual model is the result of a UNION or UNION ALL operation on one or more models and optionally the corresponding rules index.

Using a virtual model can simplify management of access privileges for semantic data. For example, assume that you have created three semantic models and one rules index based on the three models and the OWLPRIME rulebase. Without a virtual model, you must individually grant and revoke access privileges for each model and the rules index. However, if you create a virtual model that contains the three models and the rules index, you will only need to grant and revoke access privileges for the single virtual model.

Using a virtual model can also facilitate rapid updates to semantic models. For example, assume that virtual model VM1 contains model M1 and rules index R1 (that is, VM1 = M1 UNION ALL R1), and assume that semantic model M1_UPD is a copy of M1 that has been updated with additional triples and that R1_UPD is a rules index created for M1_UPD. Now, to have user queries over VM1 go to the updated model and rules index, you can redefine virtual model VM1 (that is, VM1 = M1_UPD UNION ALL R1_UPD).

To create a virtual model, use the SEM_APIS.CREATE_VIRTUAL_MODEL procedure. To drop (delete) a virtual model, use the SEM_APIS.DROP_VIRTUAL_MODEL procedure. A virtual model is dropped automatically if any of its component models, rulebases, or rules index are dropped.

To query a virtual model, specify the virtual model name in the models parameter of the SEM_MATCH table function, as shown in Example 1-4.

Example 1-4 Querying a Virtual Model

```
SELECT COUNT(protein)
  FROM TABLE (SEM_MATCH )
    '(?protein rdf:type :Protein)
     (?protein :citation ?citation)
     (?citation :author "Bairoch A.")',
    SEM_MODELS('UNIPROT_VM'),
    NULL,
    SEM_ALIASES(SEM_ALIAS('', 'http://purl.uniprot.org/core/')),
    NULL,
    NULL,
    'ALLOW_DUP=T'));
```

When you create a virtual model, an entry is created for it in the MDSYS.SEM_MODEL$ view, which is described in Table 3 above .

However, the values in several of the columns are different for virtual models as opposed to semantic models, as explained in Table 10.

Table 10 MDSYS.SEM_MODEL$ View Column Explanations for Virtual Models

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the virtual model.
MODEL_ID	NUMBER	Unique model ID number, automatically generated. Will be a negative number, to indicate that this is a virtual model.
MODEL_NAME	VARCHAR2(25)	Name of the virtual model.
TABLE_NAME	VARCHAR2(30)	Null for a virtual model.
COLUMN_NAME	VARCHAR2(30)	Null for a virtual model.
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Null for a virtual model.

Information about all virtual models is maintained in the MDSYS.SEM_VMODEL_INFO view, which has the columns shown in Table and one row for each virtual model.

Table 11 MDSYS.SEM_VMODEL_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the virtual model
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
UNIQUE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains unique triples in the virtual model, or null if the view was not created
DUPLICATE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains duplicate triples (if any) in the virtual model
STATUS	VARCHAR2(30)	Contains VALID if the associated rules index is valid, INVALID if the rules index is not valid, INCOMPLETE if the rules index is incomplete (similar to INVALID but requiring less time to re-create), INPROGRESS if the rules index is being

Column Name	Data Type	Description
		created, FAILED if a system failure occurred during the creation of the rules index, or NORIDX if no rules index is associated with the virtual model.
MODEL_COUNT	NUMBER	Number of models in the virtual model
RULEBASE_COUNT	NUMBER	Number of rulebases used for the virtual model
RULES_INDEX_COUNT	NUMBER	Number of rules indexes in the virtual model

Information about all objects (models, rulebases, and rules index) related to virtual models is maintained in the MDSYS.SEM_VMODEL_DATASETS view. This view has the columns shown in Table 11 and one row for each unique combination of values of all the columns.

Table 12 MDSYS.SEM_VMODEL_DATASETS View Columns

Column Name	Data Type	Description
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
DATA_TYPE	VARCHAR2(8)	Type of object included in the virtual model. Examples: MODEL for a semantic model, RULEBASE for a rulebase, or RULEIDX for a rules index
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

3.1.11 Semantic Data Security Considerations

The following database security considerations apply to the use of semantic data:

- When a model or rules index is created, the owner gets the SELECT privilege with the GRANT option on the associated view. Users that have the SELECT privilege on these views can perform SEM_MATCH queries against the associated model or rules index.
- When a rulebase is created, the owner gets the SELECT, INSERT, UPDATE, and DELETE privileges on the rulebase, with the GRANT option. Users that have the SELECT privilege on a rulebase can create a rules index that includes the rulebase. The INSERT, UPDATE, and DELETE privileges control which users can modify the rulebase and how they can modify it.
- To perform data manipulation language (DML) operations on a model, a user must have DML privileges for the corresponding base table.
- The creator of the base table corresponding to a model can grant privileges to other users.
- To perform data manipulation language (DML) operations on a rulebase, a user must have the appropriate privileges on the corresponding database view.
- The creator of a model can grant SELECT privileges on the corresponding database view to other users.
- A user can query only those models for which that user has SELECT privileges to the corresponding database views.
- Only the creator of a model or a rule base can drop it.

3.2.0 Semantic Annotation, Indexing, and Retrieval

Annotation, or tagging, is about attaching names, attributes, comments,

descriptions, etc. to a document or to a selected part in a text. It provides additional information (metadata) about an existing piece of data. The semantic annotation is a specific metadata generation and usage schema targeted to enable new information access methods and extend existing ones. The annotation scheme is based on the understanding that the named entities mentioned in the documents constitute important part of their semantics.

In a nutshell, Semantic Annotation is about assigning the entities in the text links to their semantic descriptions (as presented on Fig. 1 below). This sort of metadata provides both class and instance information about the entities. It is a matter of terminology whether these annotations should be called "semantic", "entity" or some other way.

Semantic Annotation helps to bridge the ambiguity of the natural language when expressing notions and their computational representation in a formal language. By telling a computer how data items are related and how these relations can be evaluated automatically, it becomes possible to process complex filter and search operations.

3.2.1 Semantic Annotation Model and Representation

Here we will discuss the structure and the representation of the semantic annotations, including the necessary knowledge and metadata. There are number of basic prerequisite for representation of semantic annotations:

· **Ontology** (or at least taxonomy) defining the entity classes. It should be possible to refer to those classes;

· **Entity identifiers** which allow those to be distinguished and linked to their semantic descriptions;
· Knowledge base with entity descriptions.

The next question considers an important choice for the representation of the annotations – "to embed or not to embed?" Although the embedded annotations seem easier to maintain, there are number of arguments providing evidence that the semantic annotations have to be decoupled from the content they refer to. One key reason is to allow dynamic, user-specific, semantic annotations – the embedded annotations become part of the content and may not change corresponding to the interest of the user or the context of usage.

Further, embedded complex annotations would have negative impact on the volume of the content and can complicate its maintenance – imagine that page with three layers of overlapping semantic annotations need to be updated preserving them consistent. Once decided that the semantic annotations has to be kept separate from the content, the next question is whether or not (or how much) to couple the annotations with the ontology and the knowledge base? It is the case that such integration seems profitable – it would be easier to keep in synch the annotations with the class and entity descriptions. However, there are at least two important problems:

· Both the cardinality and the complexity of the annotations differ from those of the entity descriptions – the annotations are simpler, but their count is usually much bigger than this of the entity descriptions. Even considering middle-sized document corpora the annotations can reach tens of millions.

Suppose 10M annotations are stored in an RDF(S) store together with 1M entity descriptions. Suppose also that each annotation and each entity description are represented with 10 statements. There is a considerable difference regarding the inference approaches and hardware capable in efficient reasoning and access to 10Mstatement repository and with 110M statement repository.

· It would be nice if the world knowledge (ontology and instance data) and the document-related metadata are kept independent. This would mean that for one and the same document different extraction, processing, or authoring methods will be able to deliver alternative metadata referring to one and the same knowledge store.

· Most important, it should be possible the ownership and the responsibility for the metadata and the knowledge to be distributed. This way, different parties can develop and maintain separately the content, the metadata, and the knowledge.

Based on the above arguments, it is proposed to decouple representation and management of the documents, the metadata (annotations) and the formal knowledge (ontologies and instance data).

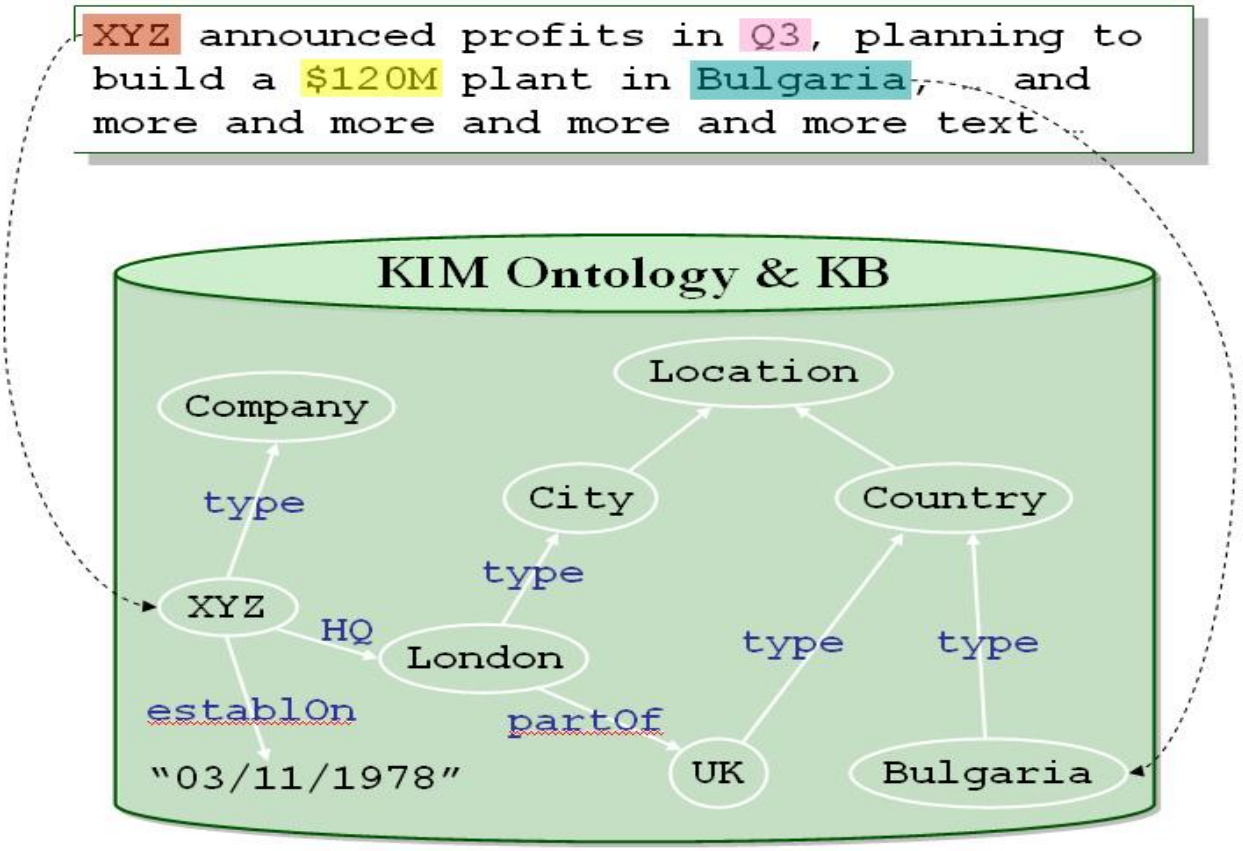

Fig. 26. Semantic Annotation

3.2.2 Semantic Annotation Process

As already mentioned, we focus mainly on the automatic semantic annotation, leaving manual annotation to approaches more related to authoring web content. Even less accurate, the automatic approaches for metadata acquisition promise scalability and without them the Semantic Web will remain mostly a vision for long time. Our experience shows that the existing state-of-the-art IE systems have the potential to automate the annotation with reasonable accuracy and performance. Although a lot of research and development contributed in the area of automatic IE so far, the lack of standards and integration with formal knowledge management systems was obscuring its usage. We claim that it is crucial to encode the extracted knowledge formally and according to well known and widely

accepted knowledge representation and metadata encoding standards. Such system should be easily extensible for domain-specific applications, providing basic means for addressing the most common entities types, their attributes, and relations.

3.2.3 Indexing and Retrieval

Historically, the issue of specific handling of the named entities was neglected by the information retrieval (IR) community, apart from some shallow handling for the purpose of Questions/Answering tasks. However, a recent large scale human interaction study on a personal content IR system of Microsoft demonstrates that, at least in some cases, the ignorance of the named entities does not match the user needs. And based on semantic annotations, efficient indexing and retrieval techniques could be developed involving explicit handling of the named entity references.

In a nutshell, the semantic annotations could be used to index both “NY” and “N.Y.” as occurrence of the specific entity “New York” like if there was just its unique ID. Because of no entity recognition involved, the present systems will index on “NY”, “N”, and “Y” which demonstrates well some of the problems with the keyword-based search engines. Given metadata indexing of the content, advanced semantic querying should be feasible.

In a query towards a repository of semantically annotated documents, it should be possible to specify entity type restrictions, name and other attribute restrictions, as well as relations between the entities of interest. For

instance, it should possible to make a query that targets all documents that refer to Persons that hold some Positions within an Organization, and also restricts the names of the entities or some of their attributes (e.g. a person's gender). Further, semantic annotations could be used to match specific references in the text to more general queries. For instance, a query such as "company 'Redwood Shores' could match documents mentioning the town and specific companies such as ORACLE and Symbian, but not the word "company".

Finally, although the above sketched enhancements look prominent, it still requires a lot of research and experiments to determine to what extent and how they could improve the existing IR systems. It is hard in a general context to predict how semantic indexing will combine with the symbolic and the statistical methods currently in use, such as the lexical approach presented in [20] and the latent semantic analysis presented in [18]. For this purpose, large scale experimental data and evaluation are required.

3.3 Entities Concept in Semantic Data Modelling

Representing the Real World with Entities in Semantic Data Modelling

In Semantic Data Modelling (SDM), an entity represents some aspect or item in the real world, such as an employee. An entity is akin to a record in a relational system or an object in an object-oriented system. These entities in SDM focus on types, which are more general, instead of sets of data. In SDM, an entity is a very basic notion of a real-world or conceptual object that is defined by a single attribute.

For instance, an SDM entity type might be person, which provides an elementary category that can be easily understood. In a relational model, however, you might end up with a number of different tables “including person, spouse, children, house, and job.” Each of these things represents part of what makes up the person, but with SDM, the person is the whole entity, rather than breaking it down into parts.

In this way, an entity in SDM is very similar to a domain. Therefore, inside this domain of person, there would be a list of names of people that are to be represented by the data. The objects in this domain would then point to specific instances of a person that are represented by each person entity. For example, the domain Person names contains Bob, Sue, Jim, Betty, and Clyde. Each of these names points to a specific object instance of Person, so that Bob points to a record giving details about Bob, such as name, gender, or marital status, and so on for each of the entities listed under Person.

3.4 Concept of Design and Indexing

Data model designs are the first and most important piece of any database application. The first logical model is about the business requirements. After the logical model is complete and approved, the physical model is materialized into a database with constraints and indexes, data types, and so on. The data model also indicates how queries can be correctly written. Joins should be used only along the relationship lines. Because there tends to be a lack of modeling discipline in some database groups, developers join on columns that seem to have the same name or whatever criteria they can come up with.

In the plant model, although there is an Area ID in the Sensor table, there is not a relationship line, so there should not be a join between those two tables. In fact, if a query is written that way, duplicate rows will be returned and a DISTINCT clause would be required. Joining all the tables results in good performance, joining numerous tables is not a problem; the problem is incorrect joining of numerous tables.

After the semantic data model is relationally correct, the next step is to add indexes for all foreign key constraints as shown in the model, and to add a single column index for each column in tables that you anticipate will end up in predicates. When there are many small indexes, the optimizer can put them together in the most appropriate manner.

It is a best practice to have the primary key be the clustered key in semantic models, unless there are good reasons not to (such as performance tests that reveal a problem with this). It is important to reiterate that there is an optimal balance between an object view and a relational view of the data and you can find this for your application only by testing.

4.0 Conclusion

Semantic data models can be very complex and until semantic databases are commonly available, the challenge remains to find the optimal balance between the pure object model and the pure relational model for each application. Semantic data models attempt to provide more powerful mechanisms for structuring objects than are typically provided by traditional approaches. Relational methods, for example, emphasize

information structures that promote efficient storage and retrieval rather than those that accommodate inter-object relationships.

5.0 Summary

In this unit we have learnt that:

- ❖ The pitfalls of Relational Language led to the development of Semantic Data Language.
- ❖ The fundamental components used by semantic models to structure data include objects, atomic and constructed types, attributes, ISA relationships, and derived schema components.
- ❖ Annotation is all about attaching names, attributes, comments, descriptions, etc. to a document or to a selected part in a text.
- ❖ The prerequisites for the representation of semantic data models are Ontology and Entity identifier.
- ❖ In Semantic Data Modelling (SDM), an entity represents some aspect or item in the real world, such as an employee.

6.0 Tutor Marked Assignment

1. (a) Explain the term data language and what led to its development.
   (b) Mention and explain the fundamental components of semantic models
2. (a) What do you understand by semantic data annotation?
   (b) Define Entity within the context of semantic data modelling

7.0 Further Reading and Other Resources

Dean M., Connolly D., van Harmelen, F., Hendler J., Horrocks I., McGuinness

D., Patel-Schneider P., Stein L.A., *Web Ontology Language (OWL) Reference Version 1.0.* W3C

Working Draft 12 Nov. 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/

Dumais S., Cutrell E., Cadiz J., Jancke G., Sarin R. and Robbins D. *Stuff I've Seen: A system for personal information retrieval and re-use.* In proc. of SIGIR'03, July 28 –August 1, 2003, Toronto, Canada, ACM Press, pp. 72-79.

Fensel D. *Ontology Language, v.2 (Welcome to OIL)* . Deliverable 2, On-To-Knowledge project, Dec 2001. http://www.ontoknowledge.org/downl/del2.pdf

Johan ter Bekke (1992). *Semantic Data Modelling*. Prentice Hall.

Module 3 Applications of Semantic Data Modelling

Unit 1 Application to Computer

1.0 Introduction

The World Wide Web started as a small Internet based hypertext project at CERN (European Organization for Nuclear Research) in late 1990. In the past ten years the Web matured to the largest distributed hypermedia system. Now, the Web with its millions of servers, pages and users constitutes the largest library of human knowledge mankind has ever created. Hypermedia systems in general and the Web in particular may be seen as a special kind of database management systems. Thus, they commit to a certain data model, at the physical, logical and semantic level.

And despite the fact that, the Web overcame these early problems it still has a number of serious limitations. Considering the Web as the largest library of human knowledge, the Web still does not support efficiently, if at all, a number of well-known knowledge transfer processes. For instance, accessing a relevant information chunk on the Web, which constitutes so-called knowledge mining process, is a rather difficult task to accomplish by means of the current Web technologies.

2.0 Objective

At the end of this unit, you should be able to:

- ❖ Explain the origin of hypertext and hypermedia
- ❖ Differentiate between Hypermedia and HC Data Models
- ❖ State and explain the classification of hypermedia systems
- ❖ Explain the application of semantic Modelling to Hypermedia

3.0 Overview of Logical Hypermedia Data Modelling

This chapter provides the overview of the most important logical data modelling paradigms in hypermedia systems in general and World Wide Web in particular.

3.1 Hypermedia systems and WWW

3.1.1 The definition of hypertext and hypermedia

As opposed to the typical printed book, where the text is read sequentially from the beginning to the end, hypertext offers a nonlinear access to the text. A typical hypertext consists of a number of chunks of textual information, usually called hyper nodes or simply nodes. Such nodes are interrelated by means of computer navigable links. Usually, links are denoted as highlighted phrases within hypertext nodes.

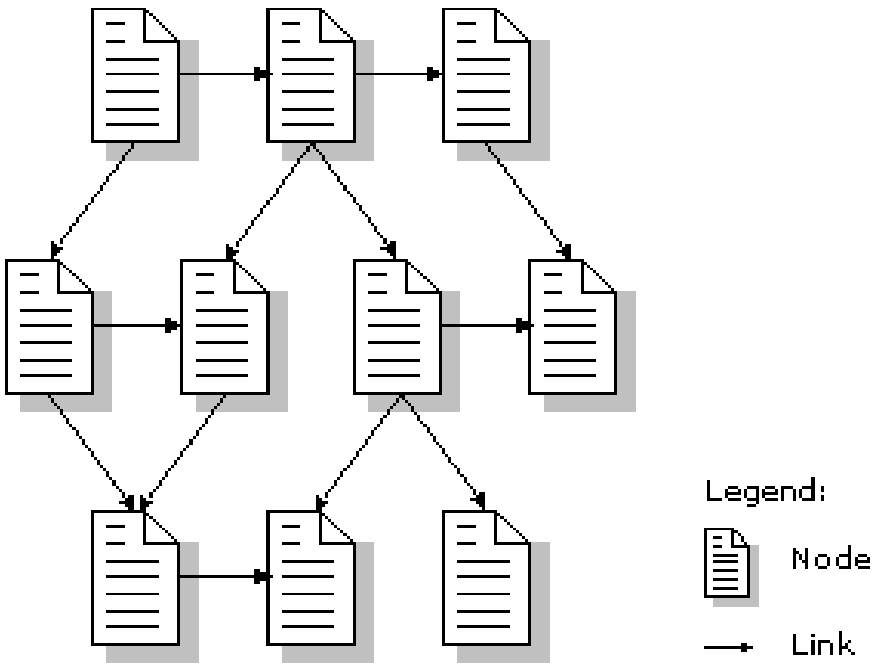

Figure 27 Hypertext

In hypertext, readers are not confronted with a prescribed reading sequence as it is the case with a book text, but rather they browse hypertext nodes activating computer navigable links. The nature of hypertext encourages readers to explore offered information intuitively, by association, following always to their most recent interests.

Hypermedia is a generalization of hypertext. In hypermedia nodes are multimedia nodes, i.e., they include not only textual information, but also other kinds of media, such as: digital images, sound clips, video clips and similar. In a sense, hypermedia combines hypertext and multimedia paradigm into a new one. Often, hypermedia is referred to as the concept of imposing a navigable structure on the top of existing collection of multimedia nodes [Maurer and Scherbakov, 1996; Maurer et al., 1998]. As Conklin (1987) states; hypermedia is a style of building systems for organizing, structuring and accessing information around a network of multimedia nodes connected together by links.

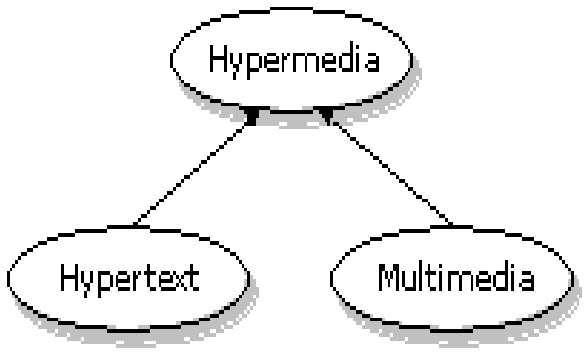

Figure 28 Hypermedia

A source anchor is the starting point of a link and specifies a part of a node where the link may be activated. Usually, source anchors are visualized in a special way (i.e., a piece of text may be highlighted) to notify users of the

existence of a link. Similarly, a destination anchor is the ending point of a link and specifies a part of a node, which should be visualized upon the link activation. Usually, an entire node is specified as the destination anchor, but a specific part of a node may be specified as the destination anchor (e.g. a paragraph within a textual node) as well.

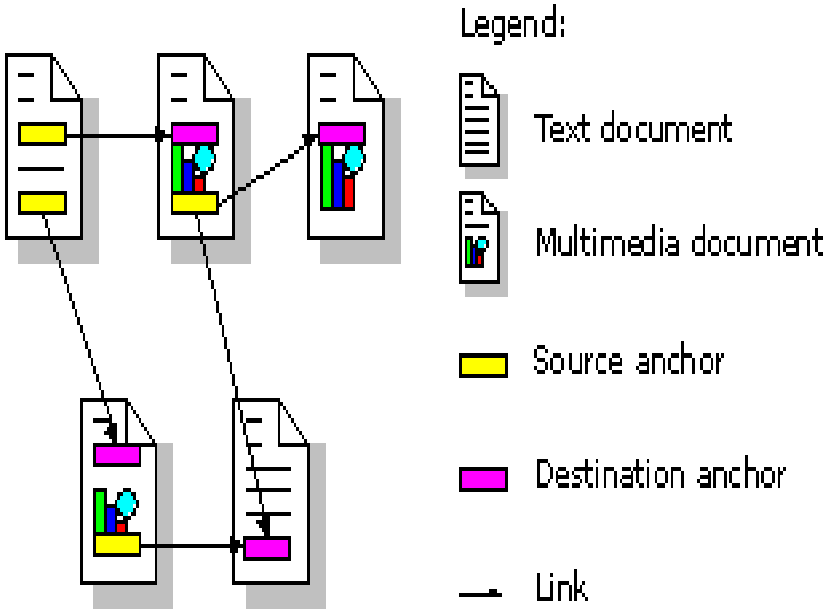

Figure 29 Documents, links and anchors in hypermedia

According to the basic structuring of data into nodes and links this basic hypermedia model is referred to as the node-link data model. Virtually, all other hypermedia models and systems may be found in this basic hypermedia data model. Likewise, a large part of hypermedia research assumes the underlying existence of this basic model [Rivlin et al., 1994]. According to this model, nodes and links in hypermedia systems form a directed graph network. Normally, such a network is called hyperweb or hyperspace.

In the early days of hypermedia development a hyperweb consisted of nodes containing mostly static multimedia information. Browsing such a hyperweb meant merely retrieving simple non-interactive multimedia documents. Recently, with the development of more advanced user interaction facilities hypermedia systems usually provide users with nodes containing interactive movies, virtual reality, collaboration and communication facilities, structured discussion forums and similar.

Hypermedia systems

We can classify hypermedia systems according to a number of criteria. For instance, considering the criteria of how much control over the form of the hypermedia presentation authors should have we get the following classification:

- Frame based hypermedia systems (all documents must fit into a fixed size frame)
- Window based hypermedia systems (documents may be of any size and they are displayed in a scrollable window area).

There are a number of other useful classification criteria for hypermedia systems. However, we consider the following classification as being of primary importance. According to the network environment in which hypermedia systems exist we distinguish between:

- Standalone hypermedia systems that provide access to a hyperweb residing on a single computer

- Large multi-user systems providing multi-point access to a hyperweb distributed over many computers connected in a network.

The focus of this work lies in the discussion of distributed hypermedia systems. A distributed environment is typically needed if either the information to be stored is too large for on machine, or if there are too many users for one machine to handle, or both. Obviously, the main prerequisite for distributed hypermedia systems is the existence of a distributed computer environment, i.e., a computer network

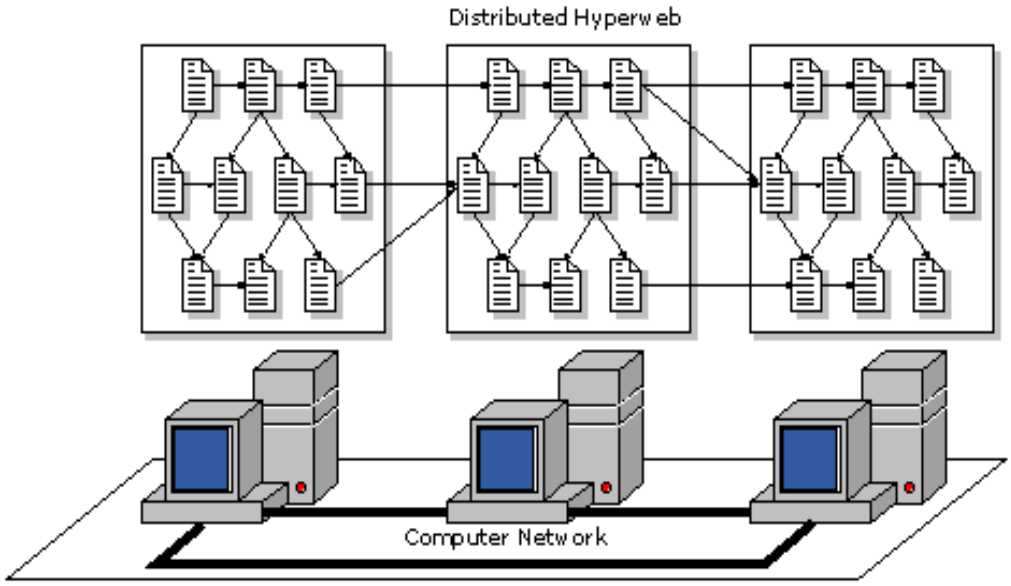

Figure 30. Distributed hypermedia systems

Nowadays, the Internet is the largest worldwide distributed computer environment existing. It is in fact a network of networks that is estimated to connect several million computers and over 50 million individual users around the world - and it is still growing rapidly.

The Internet provides its users with a number of so-called Internet services such as e-mail service, file transfer service, remote login services and similar. However, the best-known and mostly used Internet service is the World Wide Web (WWW or Web). The World Wide Web is an Internet wide distributed hypermedia system. As such it is the largest hypermedia system existing.

World Wide Web

The World Wide Web (the WWW or the Web) is the largest distributed hypermedia system nowadays. The WWW provides a remote access to the largest repository of hypermedia documents worldwide. The Web started at CERN in Geneva by Tim Berners-Lee and his team in 1991. However, the real breakthrough of the Web engaged upon the development of the first widely used Web browser, called Mosaic. Mosaic was developed by the National Center for Supercomputer Applications (NCSA). This browser provided a powerful graphical user interface following the simple point-and-click paradigm. Suddenly, by simply using Mosaic all documents residing on the Web were just a mouse-click away from users.

3.2 Logical Hypermedia Data Modelling

Web to become the most popular hypermedia system ever leading to a real explosion in numbers of Web sites and documents offered on the Web. Speaking more technically, the Web is based on typical client/server architecture. In the WWW, the whole system functionality is distributed over a tremendous number of computers that are interconnected by means

of the Internet. The WWW utilizes HTTP [HTTP, 2001] (HyperText Transfer Protocol) for client-server communication and HTML [HTML, 2001] (HyperText Mark-up Language) as a data exchange format. A particular Web server can be simply seen as a storage space for HTML and other kinds of documents where all such documents are accessible by means of so-called Uniform Resource Locator (URL) [URL, 2001].

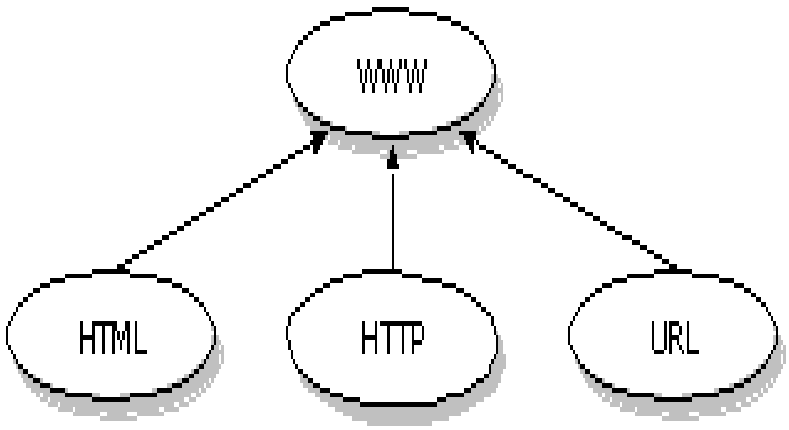

Figure 31 Basic WWW concepts

WWW servers accept HTTP requests and reply, usually, with an HTML document. Note that a URL is encapsulated into such HTTP request as a particular document identity. WWW clients just access HTML documents residing on WWW servers and visualize such documents on the user screen.

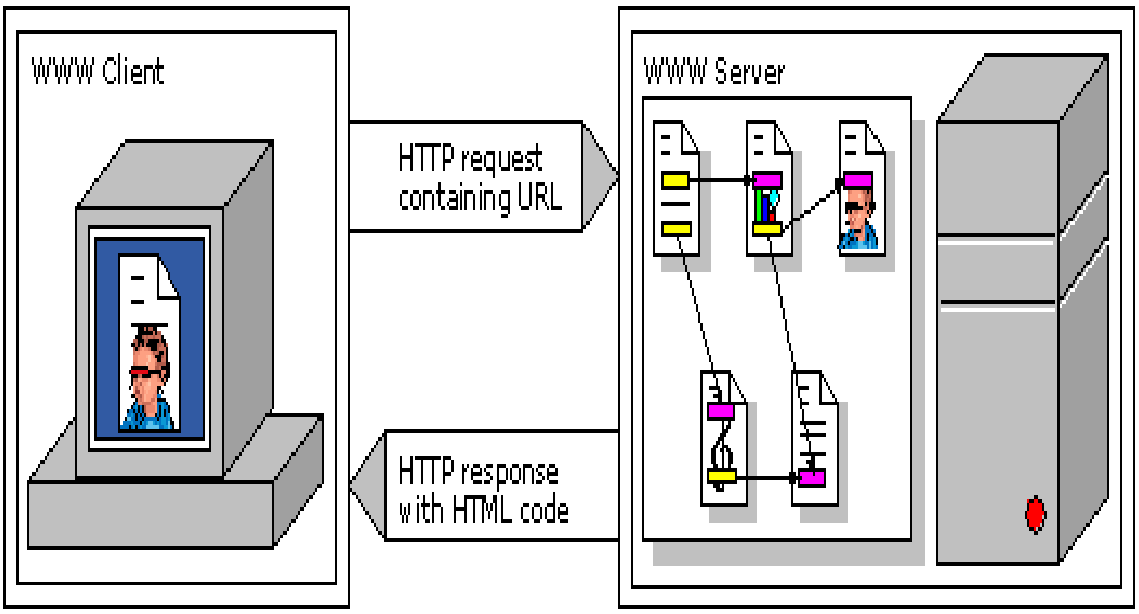

Figure 32. WWW architecture

Links in WWW are embedded into HTML documents as special tags containing a URL of a document, which is referred to. WWW clients are able to visualize and activate a hyperlink upon a user's action, thus requesting the linked HTML document to be visualized.

Alternatively to HTML documents WWW servers may store other kinds of documents such as: Windows Word [WinWord, 2001] documents, PowerPoint [PowerPoint, 2001] presentations, documents in Portable Document Format (PDF) [PDF, 2001], documents in eXtensible Markup Language (XML) [XML, 2001] format, etc. Recently, many of these document formats allow for embedding of hyperlinks, thus providing means for creating a hyperweb. However, in all such documents links are embedded into the documents, thus reducing the discussion of these formats on the discussion of the HTML format.

Semantic data model of the Web

The Web does not support the semantic level of the data modelling architecture at all. The focus of the work presented in this unit lies in defining different approaches for semantic data modelling of the Web structures. These approaches are presented in chapters to follow. However, before the presentation of those semantic data modelling approaches is given an in-depth analysis of the logical data model of the Web is required.

Current trends in logical hypermedia data modelling

In the last section a number of well-known problems of the node-link data model were presented. I listed also a number of extensions to the node-link data problem that were introduced in order to overcome these problems. In this section I provide the overview of more powerful logical data modelling paradigms and present the current trends in logical hypermedia data modelling.

As shown, the node-link data model has well-known disadvantages. However, a particular WWW server is not obliged to support the node-link paradigm internally. It can, for example, utilize the relational data model or some other data model to store data. In this case, such "advanced" WWW servers just map dynamically any incoming HTTP request into internal operations and, of course, present resultant data as HTML documents. The mapping mechanism is implemented in a so-called rendering engine running as a front-end application on a remote server [Fernandez et al., 1997; Andrews et al., 1995; Duval et al., 1995].

This approach has been successfully implemented in a number of commercial products (most notably, Home [Duval et al., 1995] and Hyper Wave).

If we compare all existing proposals for new hypermedia data modelling paradigms with the previously discussed node-link model, we may see the following main trends:

- Extending the basic hypermedia thesaurus containing notions of nodes, links and anchors with new data structures - collections, structured collections, composite components, compounds, containers or whatever. Such new data structures are addressable entities containing other data structures, nodes and links as elements [Maurer and Scherbakov, 1996; Maurer et al., 1994a, Hardmann et al., 1994, Streitz et al., 1992, Garzotto et al., 1991]. I will call such data structures composites in the further discussion.
- Providing some meta-structuring mechanism (often, referred to as templates) to predefine the structure of a number of multimedia documents, thus separating the structure from the content data [Zellweger, 1989].
- Providing multimedia documents with some attributes in addition to the content data [Maurer et al., 1998; Lennon, 1997; Andrews et al., 1995]).
- Upgrading links to independent, addressable objects separated from document content [Maurer et al., 1998; Lennon, 1997; Maurer, 1996]. - Extending the notion of an anchor by associating procedures that locate documents at run-time and even dynamically generate destination documents at run-time [Maurer, 1996].

In the next section I present the HM-Data model, a logical data model, far more powerful than the basic node-link data model that follows some of the mentioned data modelling trends.

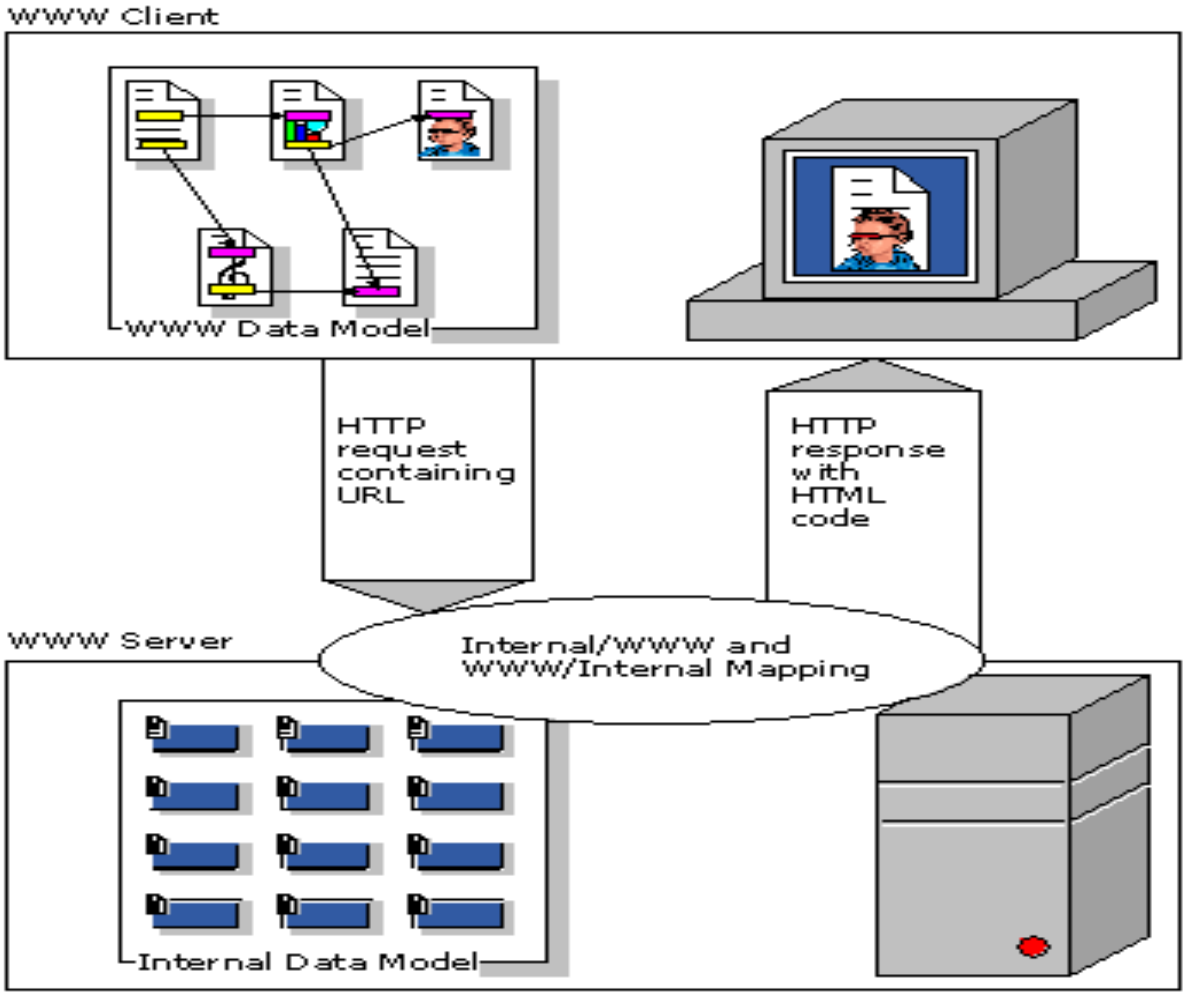

Figure 33. Mapping internal data model onto WWW data model

3.3 HM-Data model

The HM-Data Model is a logical hypermedia data model. Hence, it is less concerned with the internals of a document than it is with data structures and operations that can be applied to create, modify and explore (i.e.

render) such data structures. Therefore, in the following I will consider multimedia document as atomic, i.e. as basic indivisible chunks of multimedia data that have to be organized. I will examine the use of a novel method of hyper linking for structuring such information chunks into hypermedia databases (hyperweb).

3.4 HC-Data model

The HC-Data model is a semantic hypermedia data model. As such, it is less concerned with data structures and operations that may be applied to those data structures than it is with semantic entities and relationships that exist between these entities to which data structures may be assigned. Although the HC-Data model commits to an existence of a basic data structure, namely the hypermedia composite it still provides the possibility to model hypermedia database in the terms of types of composites (entities), members (entities) and roles (relationships between entities) of members in hypermedia composites. Even, the relationship between a member and its enclosing composite (“is-part-of” or “is-member-of”) may be seen as a relationship between the two entities. However, the HC-Data model does not allow defining other entity

An application of the HC-Data model

The HC-Data model has been successfully implemented in a novel Web-based-training (WBT) [Helic et al., 2000; Dietinger and Maurer, 1997; Dietinger and Maurer; 1998] system called WBT-Master [Helic et al., 2001;

Helic et al., 2001a; Helic et al., 2001b]. WBT-Master is an innovative WBT tool that supports the construction and delivery of Internet based courseware and provides all other important WBT information services on the base of the HC-Data model. In other words, WBT-Master is an Internet system that provides a set of modules and tools that use a unified internal data structures and well-defined set of operations applicable to such data structures.

The courseware repository on WBT-Master is structured in accordance with the HC-Data model which provides for a smooth navigation through the course eliminating problems such as "getting lost in hyperspace", dangling links and similar. The model facilitates a context-dependent search and course maps. Tutors and Learners may contribute to the courseware repository “on-the-fly” using such embedded mechanisms as annotations, links to external resources and multimedia attachments. All such additional elements may be defined as public, private or visible just to a group of people, and hence provide rather powerful customization facilities.

WBT-Master supports also more traditional communicational strategies such as discussion forums, brain storming sessions, chats, exchange with private messages (ICQ). Communication may occur between learners, tutors and groups of users. Since all the communicational tools are based on the same background – the HC-Data model, any contribution may be seen as an information object, which may be stored into a courseware repository and further reused.

Illustrative example

Consider, for instance, a hypermedia database containing a large amount of computer-based educational material (courseware), where users can browse particular courses represented in hypermedia form. To be more specific, let us assume that we have a number of computer based courses prepared in the form of HC-Units: say "Course#1" and "Course#2" that all instances of the same HC-Type, say “Course”.

Each course has a title document and a number of members corresponding to chapters of the course. If a course, let us say Course#1, refers to another course, say, Course#2, the course referred to has to be inserted into the same HC-Unit as a chapter member. A chapter consists of primitive documents and refers to other courses or chapters, i.e., a chapter is an HC-Unit of another HC-Type, say “Chapter”.
It should be noted that if particular chapters and/or documents are relevant to the contents of a number of courses, the corresponding HC-Units might be re-used without any extra overhead. HC-Unit "Author-X" has a short description about the author as its title page and the author's courses (HC-Unit "Course#1", HC-Unit "Course#2", etc.) as members, i.e., the HC-Unit “Author-X” is an instance of the “Author” HC-Type.

The above discussion has indicated how primitive chunks (i.e. documents) may be gathered into more complex structures (in this case, "chapters"). The property of the HC-Data Model that an HC-Unit may belong to many other HC-Units, even recursively, provides all the necessary power to deal with more complex situations. Note the recursive membership of documents

"Course#1" and "Author-X" in this example. Such recursive membership elegantly handles the common situation where a user needs to be able to access information about "Course#1" while browsing information about "Author-X" and vice versa. Moreover, if a certain chapter ("Chapter-B") refers to another course ("Course#3"), then the S-collection "Chapter-B" can be extended with the relevant member.

To conclude our example, the HC-Units representing courses might be inserted into HC-Units dealing with particular topics (say, "Topic#1", "Topic#2" etc., which are instances of “Topic” HC-Type). Finally, one could combine the entire topic HC-Units into an HC-Unit "Library of Courseware" of the HC-Type “Library”.

To show the basic properties of this model, let us simulate the steps of a typical user session. Suppose the user accesses the HC-Unit "Library of Courseware" in some way. Suppose that the user zooms in the HC-Unit "Library of Courseware" and activates the link to the HC-Unit "Topic#1" within the "Library of Courseware". The HC-Unit "Topic#1" becomes the current member, and the chunk of multimedia information defined as its first screen template is displayed. Note that the navigational paradigm associated with the current document "Library of Courseware" is still active. The user has the possibility to access another topic by clicking on it. After selecting a particular topic, the user can zoom into this HC-Unit. Once an HC-Unit is opened the user obtains (i.e., can follow) links encapsulated within it (perhaps a menu of courses).

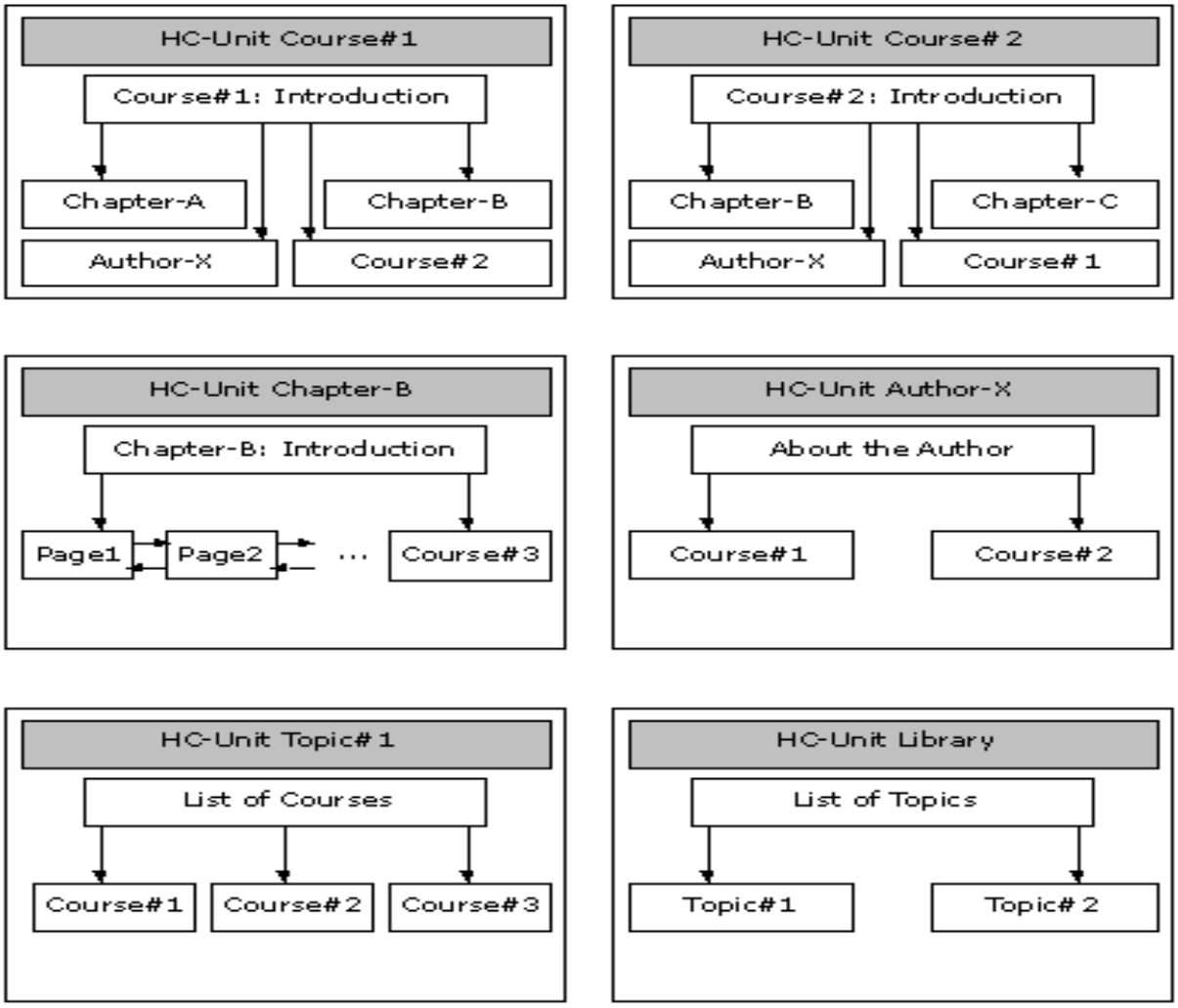

Figure 34. Sample hypermedia database

Each choice from the menu results in the presentation of the first screen template of the corresponding HC-Unit. If the user has selected and entered the HC-Unit "Course#1" and then selected the HC-Unit "Chapter-B", the starting screen template, say including the abstract of the chapter is visualized ("Chapter-B" is the current member) and links to other members (i.e., to other chapters) become available. The user can either read the current chapter (i.e., open the HC-Unit) or browse other chapters. Clicking on button “Zoom Out” returns the user to HC-Unit "Topic#1". Now

clicking on another member of "Topic#1" (say, on "Course#2") visualizes that member's starting screen template, clicking on button "Zoom Out" leaves "Topic#1", and returns the user to the HC-Unit "Library of Courseware", and so forth.

3.5 Approach Discussion of the semantic hypermedia composites

The HC-Data model is a generalization (an abstraction) of the HM-Data model in the sense that it extends the concept of hypermedia composites to the concept of semantic hypermedia composites providing a higher level of abstraction for different data structures involved in a particular application. Such generalization of the HM-Data model would not be possible without an extensive use of the Data-definition-language summarized through the concept of HC-Type, i.e., a definition of a class of hypermedia composites that all share one and the same properties and the behavior.

Hence, the HC-Data Model does not itself define a number of different classes of hypermedia composites, but rather it gives administrators facilities to define new composite types useful for a particular application.
All that is said for the HM-Data model and its advantages over the node-link data model is still valid in the case of the HC-Data model (i.e., in the case of a generalized HM-Data model). Thus, we discuss here rather the improvements of the HC-Data model over the HM-Data model itself.

The advantages of the HC-Data Model over its logical data-modelling pendant can be stated as follows:

- Possibility to apply purpose-oriented or application-specific data structures. For example, WBT-Master, being a typical WBT application, defines a wide range of data structures typical for a WBT system. Those data structures include learning units, learning course, discussion forum and similar. Such data structures are easier to apply by all users of such a system. For instance, for authors in a typical WBT system is far more intuitive to manipulate learning units or learning courses, than it would be to manipulate S-Collections.

  On the other hand, learners browse the content of such hypermedia database; the possibility to browse those data structures improves the possibility of learners to comprehend the educational material in the right way to say at least. Thus, users work with data structures that are much closer to real-life objects rather then with such generic data structures as Folders, Envelopes, etc. This facilitates improved overall understanding of the purpose of a particular hypermedia system.

- Template based authoring decreases tremendously authoring effort. The concept of the definition of a type of hypermedia composite (HC-Type) provides the base for a template-based authoring. Again, WBT-Master implements such a facility on the full extent. It allows authors a rapid production of qualitative educational material.

- Inheritance mechanism allows for creating of a wide range of instances of one end the same type. All of these instances share common properties and behaviour.
- Through the concept of properties that are defined in an HC-Type, documents are provided with useful meta-data. Those meta-data may be used to provide users with useful navigational tools such as meta-data search engines.

Such advantages provide means for supporting knowledge structuring process in hypermedia systems. Thus, a resultant hyperweb is composed of a number of well-structured knowledge chunks of hypermedia information

On the other hand, we might also see a number of disadvantages. Let us consider the following example. One of the basic concepts of the HC-Data model can be defined as imposing different types (HC-Types) of data structures on top of existing collections of HTML documents and/or other data structures. As it was the case with the HM-Data model, this allows a satisfactory level of reuse of documents and composites in different contexts.

However, all those previously discussed data structures such as hypermedia composites, different HC-Types, HC-Units were "navigational oriented" so to speak. They mainly can be perceived as different navigational paradigms, reflecting mainly different ways of accessing and working through a particular hyperweb by users of hypermedia systems. Primitively speaking, a "navigational oriented" data structure consisting of documents "B" and "C" mainly prescribes reading "B" before reading "C" and has nothing to do

with a possible situation that "C" may be a documentation on a software module implemented by the programmer "A" for a project "B". Often, users need a general overview and access to all documents provided by a particular hypermedia system.

Let us just discuss the following situation [Helic et al., 2001a]. Suppose a software organization maintains a big repository of technical documents in the form of a WWW application. Obviously, elements of the repository are valuable resources and may be reused as components of different composites. At the same time, localization of a particular document may constitute a rather difficult problem which can be only solved by structuring the repository on meta-level invariantly to any navigational paradigm. Using the same primitive language as before, we can say that the knowledge: "C" is a technical description of the software module implemented by the programmer "A" for the project "B" should be kept independently of reusing "C", "B" and "A" in different contexts.

As we see, a possibility to define richer network of entities and relationships between such entities, which is absent in the HC-Data model may be seen as being of primary importance. As already mentioned, HC-Units and their members may be seen as entities, which are related by means of the “is-part-of” relationship. Often, in order to model a particular hypermedia database on the semantic level we need to say more about the contents of this database (e.g., "C" is a technical description of the software module implemented by the programmer "A" for the project "B").

Actually, the described situation reflects the process of knowledge profiling (as described in the Section 2.10.2). Thus, the HC-Data model is not

expressive enough to provide means for creating a hyperweb with profiled knowledge attached to it.

Knowledge representation in hypermedia systems

First of all, the possibility to define new types of composites, with a possible different member roles and navigation and visualization paradigms may be seen as a great advantage of semantic hypermedia composites over their logical pendant. Such possibility provides for a resulting hypermedia database structured in accordance to the application-specific or purpose-oriented data structures, i.e., in accordance to the previously defined composite types. A hypermedia database structured in such manner is much more easily created, maintained, or accessed/browsed by a wide range of different users.

Unfortunately, a hypermedia database based on the concept of semantic hypermedia composites has also some limitations. A most important limitation of such an approach is the lack of facilities to express the semantic knowledge implicitly contained in such a database. In other words semantic hypermedia composites allow creating data structures that are highly “navigation oriented”. That means that such an approach is navigation centred, so to speak.

The navigational structure is put into the centre, either as the matter of the authoring process (i.e., the navigational structure should be easily created and maintained) or as the matter of the information retrieval process (i.e., the navigational structure should be intuitive and easily comprehended).

However, semantic hypermedia composites do not provide a possibility to structure the hypermedia database at a global semantic level. It is not possible to provide users with a general overview of the hypermedia database, i.e., with a profile of knowledge contained in such database. For instance, semantic hypermedia composites does not allow to express the fact that the document "C" is a technical description of the software module implemented by the programmer "A" for the project "B", but merely they prescribe that the document “B” should be read before the document “C”.

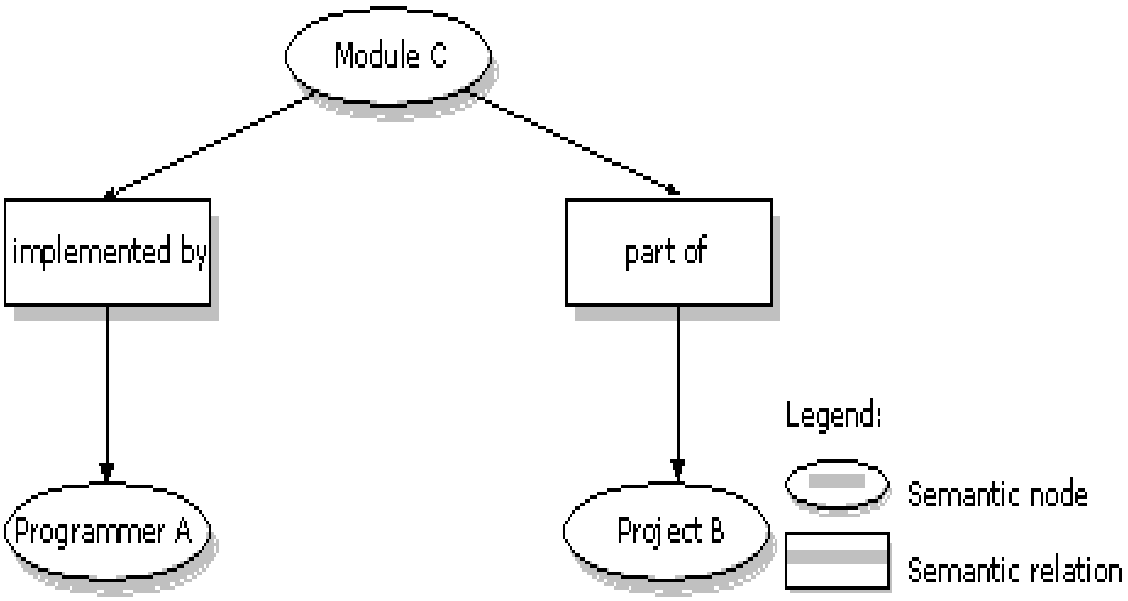

Figure 35. Semantic network of resources in a hypermedia database

Properties of semantic networks

A hypermedia database structured accordingly to a knowledge domain schema is a repository of well-structured reusable hypermedia modules enhanced with a semantic network of those modules. The semantic network

represents knowledge contained in those hypermedia modules in the form of different concepts, to which hypermedia modules may be assigned, and relationships between these concepts.

Generally, semantic networks may be used as a component of two different information-processing tasks:

- **Information Retrieval:**
    - o User Interface: Semantic network may be seen as a simple and intuitive visual form of knowledge representation and, thus, as a metaphor for man-machine communication [Sowa, 1984; Sowa, 1991].
    - o Querying: Semantic network may be seen as a database schema that allows for querying the concepts and their relationships [Sowa, 1984; Sowa, 1991].
- **Inference Engine:** Semantic network may be seen as a special formalism for defining rules of inference [Sowa, 1984; Sowa, 1991].

A Knowledge Domain apply a semantic network to create a convenient and intuitive graphical user interface for browsing the hypermedia database, as well as a powerful search mechanism to search for hypermedia modules represented by concepts from a particular semantic network.

However, Knowledge Domains does not make use of the other very important property of semantic networks, i.e., the possibility to infer a new knowledge from the semantic network by an application of so-called inference rules. The concept of knowledge cards tries to make use of the inference power of semantic networks. Furthermore, this concept tries to combine the information retrieval facilities of semantic networks with an inference engine based on a number of very simple inference rules.

Such a global semantic structuring may be accomplished only by means of more powerful knowledge representation mechanisms, which include a conceptual structuring of the subject matter. Such mechanisms usually support the definition of so-called domain ontology, which defines concepts, attributes and relationships between such concepts in accordance with a particular problem domain.

The hypermedia database is structured corresponding to a number of such definitions, where particular document and/or composites from the hypermedia database are assigned to an arbitrary number of concepts and relationships. The outcome of such structuring technique might be for instance represented by means of network knowledge representations in the form of a semantic network of hypermedia nodes containing a semantic knowledge about hypermedia nodes and relationships between such nodes.

Such graphical representations [Gains and Shaw, 1995] facilitate greatly information retrieval techniques usually applied in hypermedia systems: browsing [Gains and Shaw, 1995; Brusilowski and Schwarz, 1997; Barwise and Etchemenedy, 1990; Chang et al., 1986]. Of course, querying of such data structures [Arents and Bogaerts, 1996; Comai et al., 1998] might be also easily facilitated.

3.6 Introducing Semantic Data Structures to the WWW

The semantic hypermedia data modelling concepts that were introduced in the last chapter were successfully implemented in the novel Web-based-training system called WBT-Master [Helic et al., 2001; Helic et al., 2001a;

Helic et al., 2001b]. This chapter gives an overview of implementation issues of those data modelling approaches.

3.7 WBT-Master

The WBT-Master is a novel Web-based-training system implemented on the following concept: a modern WBT system should provide a set of tools which support different knowledge transfer processes, thus allowing for a smooth transfer of knowledge from people who posses such knowledge (e.g. experts, teachers) to people who want to acquire this knowledge (e.g. learners, students) [Helic et al., 2001b]. Thus, WBT-Master provide tools that support the following knowledge transfer processes in a Web based environment [Helic et al., 2001b]:

- Web based knowledge structuring
- Web based knowledge mining
- Web based knowledge profiling
- Web based tutoring
- Web based mentoring
- Web based learning.

Thus, WBT-Master was an implementation of the previously discussed knowledge transfer processes in a Web environment. Here we provide the overview of such implementation.

WBT-Master architecture

Being a fully WWW compatible, WBT-Master considerably extends the standard WWW client-server architecture. WBT-Master servers store complex, composite data objects additionally to primitive HTML documents. The data objects are typed, i.e. a data objects always belongs to a particular data class, which defines all user's actions applicable to such data objects. Documents, portals, learning units, learning courses, learning goals, etc. are data objects residing on WBT-Master server.

The data objects are persistent, i.e. a WBT-Master server can apply so-called actions, which alter a current state of a data object, and results of such actions are available to other users. For example, a new member may be inserted into a learning unit, a new contribution added to forum, etc.

The data objects are reactive, i.e. an object replies to an action with a collection of data which depends on the current state of the data object, and user's context. For example, "get content" action addressed to a discussion forum, returns a structured collection of contributions made to this forum. Actions are sent from an Internet client to the WBT-Master using ordinary HTTP protocol. Note that the HTTP requests contain not only the URL of a particular data object but also an action, which is to be applied to this data object. The WBT-Master server carries out the request, which results in:

- Possible modification of the target object current state
- Generating a response to the action.

The server response is visualized on the WBT-Master client side as resultant information and all further actions, which can be applied to such data object (as opposed to a visualization of passive HTML documents).

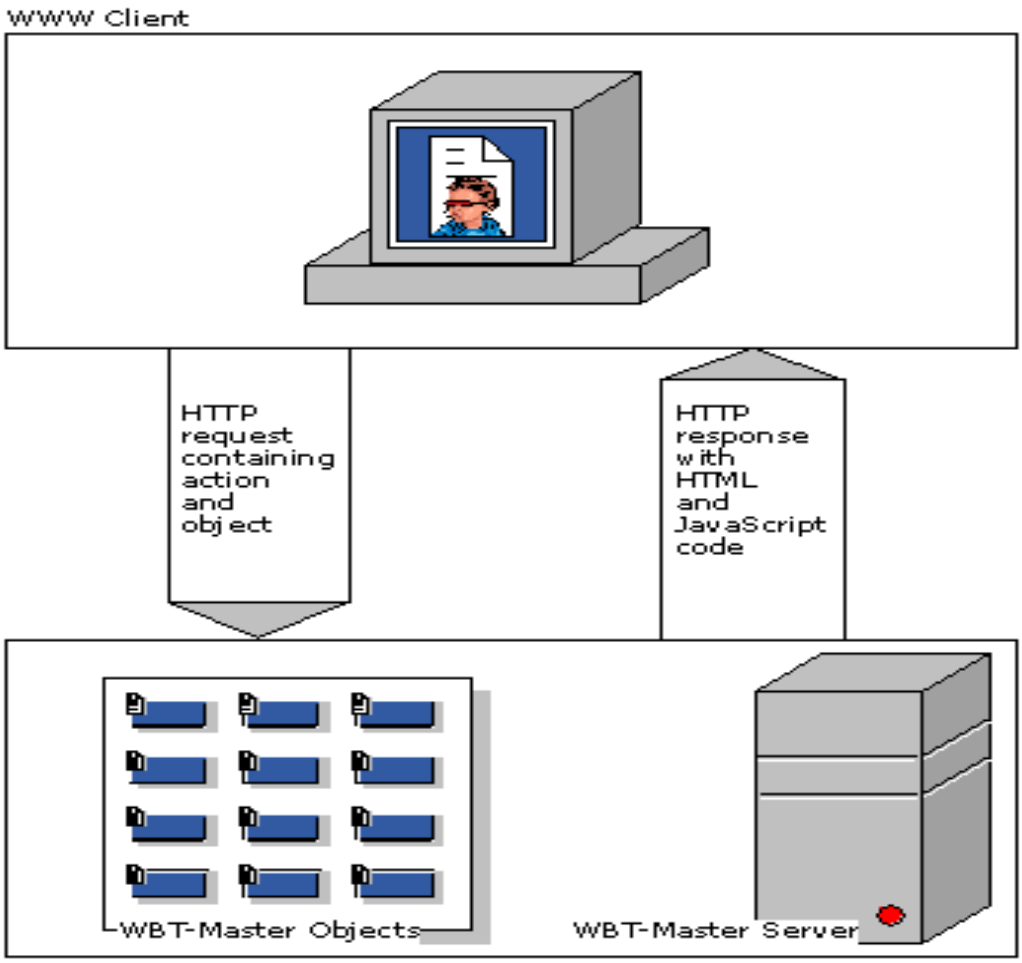

Figure 36. World Wide Web Client

All the previously discussed data structures, such as Knowledge Domain, Knowledge Cards or HC-Types and HC-Units are different kinds of WBT-

Master objects. Thus, WBT-Master provides an implementation of the previously discussed semantic data modelling approaches.

WBT-Master technical solutions

Recollect the basic WBT-Master architectural principles [WBT-Master, 2001]:

- Data Objects are persistent, reside on a server and are eligible for actions that may alter the data objects.
- A particular data object belongs to one of predefined data types that define properties of the data object and valid actions.
- A client initiates an HTTP request containing reference to a target data object, an action that need to be applied to it and a number of parameters relevant to this action.
- A server applies the action to a prescribed data object. The action may affect the object's current state. The response to the action is sent back to the client where it is visualized on the user's screen.

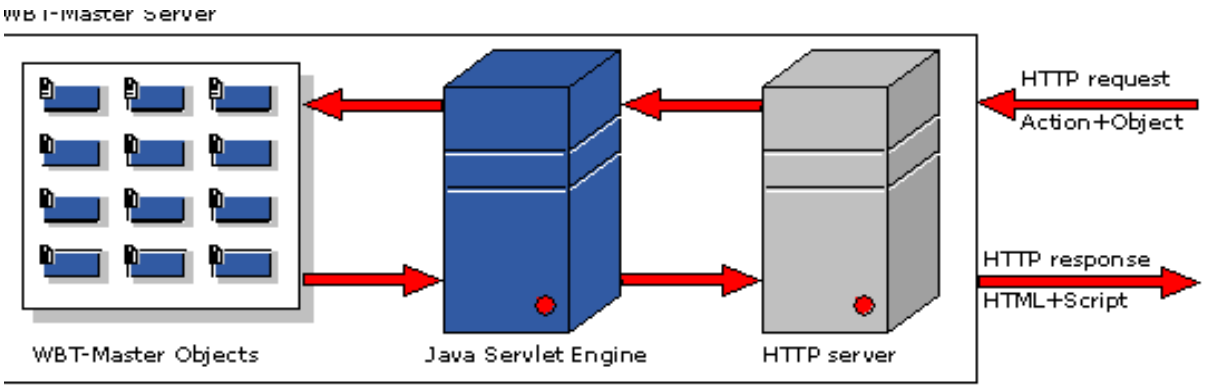

Figure 37. WBT-Master server architecture

Obviously, standard WWW servers and clients do not support the above-mentioned functionality. Hence, the functionality of both components (i.e. server and client) must be extended by means of one of modern WWW

programming technologies. On the server side WBT-Master extends a standard WWW server functionality by applying so-called Java Servlet (a Java-enabled Web server). Since Java Servlet are small Java programs running in a context of a Java Servlet Engine, they provide an ideal mechanism for implementing WBT-Master actions.

Current implementation of the WBT-Master is based on Apache Web Server with the Apache Servlet module enabled and JServ Servlet Engine. However, any server that supports server site Java Applets (i.e. Servlets) can be used as well. An Apache Web Server acts as a repository of data objects. It stores data objects in its file system. A client request including an encapsulated action is interpreted by a Servlet which actually generates a response depending on the current object state.

The current Servlets implementation assumes that data objects are instances of an abstract Java class. This basic abstract class is defined through its method interface that represents actions that can be applied to data objects. Subclasses of the basic abstract data object class support particular logic of an action applied to a predefined type of a data object.

This level of abstraction enables dynamical mapping of logical data structures onto a variety of possible physical storage spaces. For example, an ordinary Apache file system may be replaced with a database by implementing a new subclass of the basic abstract class.

Moreover, since the servlets utilize the Java Factory concept it is possible to change the physical data objects format even at the runtime.

Conceptually, an end-user always communicates with a particular instance of data object. Thus, for example, the user may work on a particular Learning course, Learning goal, Forum, etc.

On the client side, JavaScript functions and signed Java applets are used to:

- Visualize all actions applicable to a current data object;
- Convert a particularly selected action into an HTTP request;
- Visualize the action's results (i.e. server response) on the user's screen.

4.0 Conclusion

With exploration of the application of semantic data modelling to Computer world, it is now clear that, interconnected information is fundamental to hypermedia. So, semantic data models are particularly effective in capturing the complex inter-object relationships that characterize hypermedia; its techniques also allow structure, information, and behaviour to be abstracted from hypermedia.

5.0 Summary

In this unit we have learnt that:

- ❖ World wide web started as a small Internet based hypertext Project at CERN (European Organisation for Nuclear research) in late 1990.
- ❖ A typical hypertext consists of a number of chunks of textual information, usually in hypernodes or simply nodes.

- ❖ Hypermedia systems can be classified as Frame-based, window-based, Standalone and Large multi-user system.
- ❖ The HC Data Model is a generalisation (abstraction) of the HM Data Model, in the sense that, it extends the concept of hypermedia composites to the concept of semantic hypermedia.

6.0 Tutor Marked Assignment

1. (a) What do you understand by the term Hypermedia?
   (b) Mention and explain the classification of hypermedia
2. (a) Differentiate between Hypermedia and HC Data Models
   (b) Explain the application of Semantic Modelling in hypermedia

7.0 Further Reading and Other Resources

[Al-Khatib et al., 1999] W. Al-Khatib, Y. Francis Day, A. Ghafoor, P. B. Berra: Semantic modeling and knowledge representation in multimedia databases, IEEE Transactions on Knowledge and Data Engineering, 11(1), 64-80, 1999.

[Andrews et al., 1995] K. Andrews, A. Nedoumov, N. Scherbakov: Embedding Courseware Into Internet: Problems and Solutions, Proceedings of ED- MEDIA'95, Graz, Austria, 69-74, 1995

[Andrews, 1996] K. Andrews: Browsing, Building and Beholding Cyberspace: New Approaches to the Navigation, Construction and Visualization of Hypermedia on the Internet, PhD. Thesis, Graz University of Technology, Austria, 1996.

[Arents and Bogaerts, 1996] H.C. Arents, W. F. L. Bogaerts: Concept-Based Indexing and Retrieval of Hypermedia Information, Encyclopedia of Library and Information Sciences (supplement volumes), Vol. 58, 1-29, Academic Press, New York, 1996.

[ASP, 2001] Active Server Pages (General), http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000440

[Barwise and Etchemenedy, 1990] J. Barwise, J. Etchemenedy: Valid Inference and Visual Representation, Zimmerman and Cunningham (Ed.), Visualization in Mathematics, American Mathematics Association, 1990.

[Berners-Lee et al., 1992] T. Berners-Lee, R. Cailliau, J. F. Groff, B. Pollermann: World-Wide Web: The Information Universe, Electronic Networking, Research, Applications and Policy 1(2), 74-82, 1992.

Module 3 Applications of Semantic Data Modelling

Unit 2 Business Semantics Management for effective Application Integration

1.0 Introduction

In today's business ecosystems, information has become a competitive and strategic asset. Being able to exchange data and to interpret this data in the right context and within a reasonable time is a top priority for many organizations. And adding semantics and business context to your integration architecture will reduce complexity, increase agility, and improve governance. This integration model is considered a best-practice when integrating applications on an Enterprise Service Bus.

The challenges of implementing this architecture successfully are threefold:

(1) Lack of semantic alignment,

(2) Lack of flexibility, and

(3) Lack of governance.

These challenges are discussed in details, and answers are provided on how to tackle these issues by:

(1) Adding business context to your disparate data sources and let it drive the integration process,

(2) Involving technical and business stakeholders by de-coupling structure from meaning in terms of business vocabularies, facts and rules.

(3) Leveraging these business semantics operationally by deploying them as data services on your Enterprise Service Bus.

2.0 Objective

At the end of this unit, you should be able to:

- ❖State the challenges of sharing data models

- ❖ Describe the business semantic for application integration
- ❖ Describe the business semantics management product suite

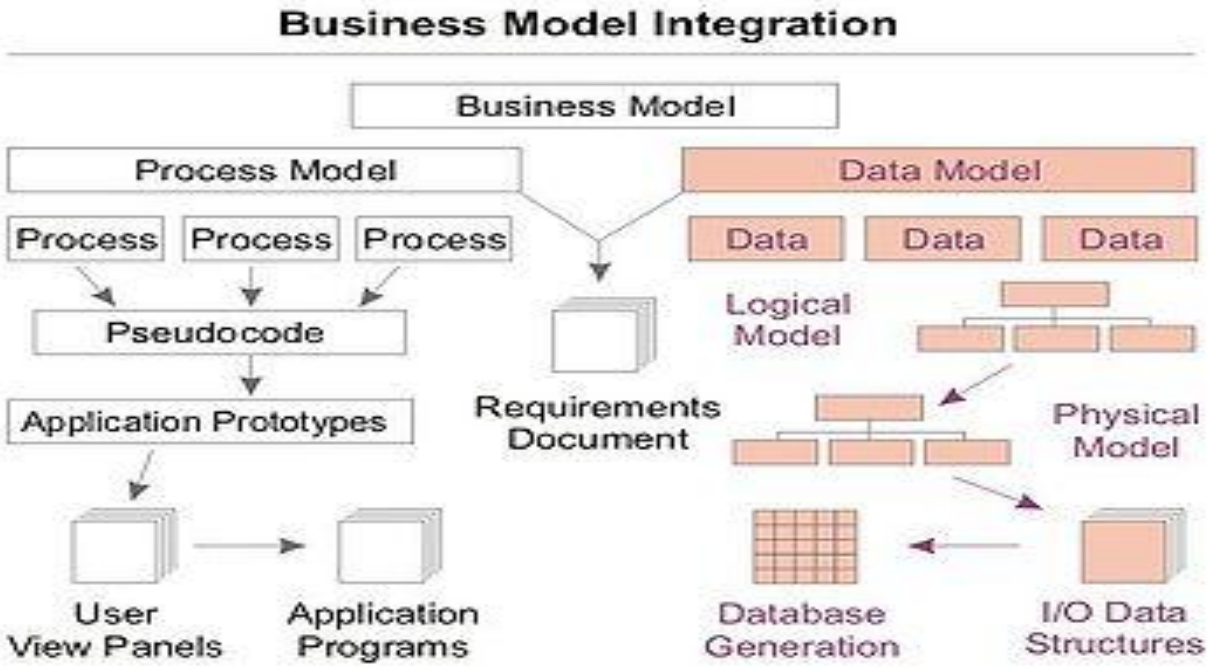

Figure 40. Business Model Integration

3.0 Challenges of a shared data model

With these ambitious goals, in a complex environment as application integration, it is quite obvious that achieving these benefits comes with quite some challenges. Three gaps have been identified in current solutions that will need to be overcome: Lack of semantic alignment, lack of flexibility, lack of governance.

Lack of semantic alignment

From a conceptual and business perspective, the structure and the meaning (read: semantics) of a data model are completely unrelated. Data models are usually created in XML or UML. These Models have a well-defined structure and relations, but do not sufficiently define the meaning of the information concepts. Furthermore, they are attributed the status shared because, they are modelled in a standard format, and not necessarily because they are the product of an agreement in a community of practice. If we take XML as an example, the following quote describes the limitations of XML as a means to capture the semantics of data:

"While the definition of an XML protocol element using a validity formalism is useful, it is not sufficient. XML by itself does not supply semantics. ...

Any document defining a protocol element with XML MUST also have sufficient prose in the document describing the semantics of whatever XML the document has elected to define."RFC 3470, "Guidelines for the Use of XML within IETF Protocols" January 2003 Even if we assume a model is shared, a merely technical data model does not sufficiently grasp the contextual meaning of business assets in the organization. These results in semantic gaps over three different dimensions: from systems to systems, from people to people, and from people to systems. A shared data model in XML or UML covers none of these gaps:

Systems to Systems: While the disparate systems can be connected through a shared, but technical, data model, that model does not supply any

semantics. The semantic gap between these systems can therefore not be identified nor solved.

People to Systems: Given the lack of semantics in such a shared data model, it is still subject to interpretation from the different stakeholders. A shared data model does not reduce the risk of interpretation errors leading to erroneous transformation mappings, model use, etc.

People to People: By not making the interpretation of the different data formats explicit, the traditional misalignment between different stakeholders (business and IT) is not solved. This leads to increased project costs through miscommunication and repeated work.
“The market lacks tools capable of rationalizing business process models, logical information models and repository management tools for automated semantic resolution in SOA.”

Lack of fexibility

The flexibility of your shared data model is crucial to cope with what we call the coverage of your disparate systems. The goal should be to have a shared model that covers 100% of the systems to be integrated. The following illustration provides an example:

Lack of governance

When your shared data model is polluted by application-specific structure and does not provide the necessary domain semantics, it remains on a

technical level. This is sufficient for basic governance like impact analysis. But it goes much further than that. The goal should be to push the shared data model to the same level as business process management tries to do with processes. To keep this shared data model alive and valuable for the organization, it is imperative to involve all its stakeholders: the domain experts (usually the business (analysts)), the functional analysts, and the technical experts. If you involve business stakeholders, they will want to be able to have a say in the process.

Assigning concept stewards is a great way to involve these users and keep the shared data model alive and up-to-date with the ever-changing business requirements. Concept stewardship, however, doesn't work well on models defined on a technical level. Another important governance aspect is compliancy with industry standards. When directly using one standard, you are technically bound to this inflexible, static standard. This makes compliance with other standards, or adding new concepts and relations to your shared data model, technically and operationally complex. Developing a semantic model from a relevant business context perspective instead of an application or standard perspective, will add a valuable de-coupling layer that will provide compliancy with industry standards and remain flexible and efficient in its use.

3.1 Business Semantics for Application Integration

Not acting on the three challenges identified before (lack of semantic alignment, lack of flexibility, lack of governance), will not only increase the

risk of project failure, it also limits the potential up-side of your investment. Adopting a shared information model is a great opportunity to better align business & IT, increase information governance through transparency and traceability, and create a true enterprise asset, without additional costs. The vision is to help companies achieve semantic intelligence.

Similar to Operational Intelligence or Business Process Intelligence, which aims to identify, detect and then optimize business processes, semantic intelligence targets information instead of processes. It aims to enable better understanding and provide insight in data for all stakeholders. Semantic Intelligence supports better information sharing, reuse, governance and better decision-making.
What we will describe here is how taking steps towards that goal will solve the three major challenges of adopting a shared information model for application integration. The core of this approach is to separate the meaning from the structure of the shared information model. Doing so enables all stakeholders to understand the shared model, increases flexibility through additional de-coupling, and promotes governance and manageability by enabling each stakeholder to govern its own interest. With respect to metadata, four different levels can be identified, namely the conceptual-, relational-, technical-, and operational level.

The conceptual layer describes the business concepts in the organization, understandable for any stakeholder familiar with the domain. In practice, these are often stored in dictionaries, vocabularies, and taxonomies.

A second layer is the relational layer where facts, relations and patterns of these business concepts are managed. These help the business to put business concepts into context, see how these concepts are used, and how they relate to other concepts. A third level is the technical level. This level describes the concepts and relations from the two levels above, but in an application-specific way. Typical standards used on this level are UML and XSD (XML Schema). These data models consist of sufficient application semantics to be directly linked to applications or to be used for model-driven development. The operational level is where the actual data sources are located.

Currently, metadata on all four of these levels is mostly managed in an ad-hoc manner. Within a certain layer, there is often no linking, harmonization, reuse or reconciliation between disparate metadata models. The only place where this does occur is on the technical level, using metadata repositories (usually several disconnected ones). These repositories however, are very tool-dependent and fail to reach sufficient coverage. They don’t harmonize nor reconcile metadata on the other levels. Interestingly, the more metadata moves from a technical to a business context, the more it becomes ad-hoc. Business people usually use typical office tools like word processors, document managing systems or spreadsheets to manage business-critical metadata.

3.1.1 Semantic Alignment

It is imperative to push the shared information model up to the relational and conceptual levels. Doing so will enable you to de-couple the application-specific structure of your data model from the meaning of the business concepts. On the conceptual level, it enables the business to collaboratively capture and define the business concepts relevant for the problem domain through simple definitions, examples, synonyms, etc.

On the relational level, it enables the business and technical stakeholders to think in a fact-based manner how these business concepts relate to each other. These business facts can be grouped together into a semantic pattern which captures the semantics of a certain problem scope on which all stakeholders can easily agree. This semantic pattern is, in this case, the conceptual representation of your shared information model. Technical models such as UML, XSD or database schema's can be automatically generated from a semantic pattern through an automatically generated commitment.

The approach illustrated above, enables you to cover all four metadata levels. This in turn will help you to involve all stakeholders in the process. Because all stakeholders are involved, the result will be of higher quality and will remain up-to-date from a business as well as a technical perspective.

3.1.2 Flexibility

The core flexibility problem in adopting a shared information model lies in the big impact of necessary changes to the shared technical model on all the other existing applications. On the one hand, it should be possible to easily change the shared model as needed to get the new integration requirements finished in time. On the other hand, the shared model should be well governed or it will explode in size and complexity, and will quickly become so unpractical it becomes just another data format to integrate. The flexibility to change the shared information model as needed, and the ability to govern these changes effectively.

Key in this approach the point where the reconciliation process and the application process come together: The Unify activity. Collibra's Business Semantics Management product suite features a feedback loop through Collibra's Business Semantics Enabler product.

3.1.3 Governance

When adopting a shared information model for application integration, being able to effectively govern that model is crucial to the project's success. Collibra introduces a closed-loop feedback technique to make sure all different stakeholders can act effectively on their responsibilities:

- The technical stakeholders can make the necessary changes when the shared model is insuffcient to solve the integration problem.
- The business stakeholders and domain experts can make sure the shared model correctly describes the business context.

The business stakeholders get involved by appointing concepts stewards to govern the concepts described on a business level. The technical stakeholders create and govern the commitments between the business context (the semantic pattern) and the underlying applications. These commitments are pushed up to the business level through a closed-loop feedback mechanism. Collibra's patented technology enables all stakeholders to collaboratively govern the changes in these semantic patterns. This provides the necessary versioning, transparency and traceability functionality.

3.2 Example from the supply chain industry

As a practical example, we present an implementation of Business Semantics for application integration at SCA Packaging. SCA Packaging is a leading European provider of customer-pecific packaging solutions with 220 production units in 30 countries and 15 500 employees. SCAi is SCA's integration platform as well as a competence centre, across all SCA Packaging divisions and regions, dedicated to maintaining a prompt and cost-effective response for all integration needs. Some figures on the current SCAi setup: 400 different message formats, 600 mappings since start up, 3,500 routes, approximately 400 nodes, a core team of 6 FTEs. SCA Packaging is a member of the papiNet standard group, a global community involved in supply chain processes for the forest and paper industries. The papiNet group has developed a standard XML schema that should represent the business concepts in this industry. However, as we have described above, this standard quickly becomes just one of the systems to be integrated, next to the

hundreds of EDI, XML or database systems.

semantic data integration solution enables SCA to:

Reduce the complexity and number of mappings by committing to a shared business context / semantic model. Increase the flexibility of the solution by adding a de-coupling layer between different standards and message formats. Increase efficiency by allowing people to effectively communicate and agree on the meaning of the business context and reuse these concepts in different integration scenarios. Increase agility and governance by involving domain experts (usually business analysts or enterprise architects) to define and govern the business context which drives the integration process.

The different applications are no longer integrated by ad-hoc, point-to-point mappings but through a shared semantic model which describes the business context. It also shows how this additional de-coupling supports different and changing standards or pre-existing internal data models in a technical format such as XSD or UML.

Furthermore, the three transparent rectangles show how the process of agreeing on the meaning of the business assets in their business context can be effectively achieved by domain experts without the additional application-specific format complexity. The technical experts doing the actual integration have a hard time agreeing on a shared meaning because they think in terms of their application-specifc data formats. Using our approach, they can simply connect, or what we call commit their application-specifc formats onto the shared business context.

Collibra Software

Collibra is an enterprise software company integrating with and adding Business Semantics to all major SOA, EAI & Data integration vendors. Business Semantics Management defines the specific meaning and context of key business assets (e.g. ‘ customer’, ‘product’, ‘interest rate’,...) for organizations, converting them into clearly defined business facts and rules compiled in a ‘Business Glossary’.

The solution translates these key business assets into executable models that drive EAI/SOA, Business Intelligence, Semantic Data Integration and Metadata based on relevant business context instead of pure system to system integrations.

3.4 Business Semantics Management product suite

Business Semantics Glossary

The Business Semantics Glossary is a web-based product aimed at both business as well as technical users. It lets people collaboratively define and govern the meaning of the business assets in their business context. It is the first enterprise-ready tool that implements the OMG’s SBVR industry standard (Semantics of Business Vocabulary and Business Rules) to define business vocabularies, facts and rules.

Business Semantics Studio

The Business Semantics Studio is an Eclipse-based tool suite that enables IT professionals to tie meaningful business context to the technical models

and (legacy) data sources. It provides modelling functionality to extract semantic patterns from a business context. It also provides mapping functionality to commit existing data sources onto these semantic patterns.

Business Semantics Enabler

The Enabler is a run-time server that leverages your IT infrastructure by generating data services based on shared contextual meaning. The Information Enabler enables automatic semantic integration possible: given a set of business semantics, the Information Enabler can automatically trans-form data between different data formats or web-services.

4.0 Conclusion

Being able to exchange data and to interpret the information in the data that has been exchanged in the right context and within a reasonable time is a top priority for many organizations. In this white paper, we have identified three critical challenges when adoption a shared data model to drive application integration:

(1) Lack of semantic alignment: Technical data models do not provide the business context nor the semantics of the data. It does not solve the semantic gaps between people, between systems, and from systems to people.

(2) Lack of flexibility: A technical data model is application specific. This makes it extremely complex and inefficient to cover all the semantic differences of the different applications in a single data model or manage different shared data models that cover these semantic differences.

(3) Lack of governance: A shared model should involve all its stakeholders, business and IT. The business context that drives the integration should be well governed and kept alive. In this white paper, we have laid out several solutions to cope with these challenges:

(a) Adding business context to your disparate data sources and let it drive the integration process,

(b) Involving technical and business stakeholders by de-coupling structure from meaning in terms of business vocabularies, facts and rules.

5.0 Summary

Many organizations have chosen to adopt a shared or so-called canonical data model for their data and application integration. And that, such an integration pattern, in theory, would increase the efficiency, agility and transparency of application integration for the organization. As a result, adding business context and semantics to the existing infrastructure in this manner, enables organisations to leverage investments by increasing agility and governance and reducing complexity.

6.0 Tutor Marked Assignment

1. (a) What are the challenges of sharing data model
   (b) Mention some of the Business Semantics Management product suite
2. What do you understand by the term Flexibility in Business Semantic Management.

7.0 Further Reading and Other Resources

www.collibra.com tangible solutions for your business semantics

R. Garcia and O. Celma. Semantic integration and retrieval of multimedia metadata. In Proc. of the 5th International Workshop on Knowledge Markup and SemanticAnnotation (SemAnnot 2005), Galway, Ireland, November 2005.

Potter, W. D., and Trueblood, R. P. Traditional, semantic, and hyper-semantic approaches todata modelling. Computer 21, 6 (June 1988), pp. 53−63.

Peckham, J., and Maryanski, F. Semantic data models. *ACM Comput. Surv.* 20, 3 (Sept.1988), 153−189.

Hammer, M., and McLeod, D. Database description with SDM: A semantic database model.

ACM Trans. Database Syst. 6, 3 (Sept. 1981), 351−386. 25. Hudson, S. E., and King, R. The Cactis project: Database support for software environments. *IEEE Trans. Softw. Eng.* 14, 6 (June 1988), 709−719.