Knowledge Engineering

Kaylah Broadway

First Edition, 2012

ISBN 978-81-323-3481-1

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter-1

Knowledge Engineering

Knowledge engineering (KE) was defined in 1983 by Edward Feigenbaum, and Pamela McCorduck as follows:

KE is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise.

At present, it refers to the building, maintaining and development of knowledge-based systems. It has a great deal in common with software engineering, and is used in many computer science domains such as artificial intelligence, including databases, data mining, expert systems, decision support systems and geographic information systems. Knowledge engineering is also related to mathematical logic, as well as strongly involved in cognitive science and socio-cognitive engineering where the knowledge is produced by socio-cognitive aggregates (mainly humans) and is structured according to our understanding of how human reasoning and logic works.

Various activities of KE specific for the development of a knowledge-based system:

- Assessment of the problem
- Development of a knowledge-based system shell/structure
- Acquisition and structuring of the related *information*, *knowledge* and specific *preferences* (IPK model)
- Implementation of the structured knowledge into knowledge bases
- Testing and validation of the inserted knowledge
- Integration and maintenance of the system
- Revision and evaluation of the system.

Being still more art than engineering, KE is not as neat as the above list in practice. The phases overlap, the process might be iterative, and many challenges could appear. Recently, emerges meta-knowledge engineering as a new formal systemic approach to the development of a unified knowledge and intelligence theory.

Knowledge engineering principles

Since the mid-1980s, knowledge engineers have developed a number of principles, methods and tools that have considerably improved the process of knowledge acquisition and ordering. Some of the key principles are summarized as follows:

- Knowledge engineers acknowledge that there are different types of knowledge, and that the right approach and technique should be used for the knowledge required.
- Knowledge engineers acknowledge that there are different types of experts and expertise, such that methods should be chosen appropriately.
- Knowledge engineers recognize that there are different ways of representing knowledge, which can aid the acquisition, validation and re-use of knowledge.
- Knowledge engineers recognize that there are different ways of using knowledge, so that the acquisition process can be guided by the project aims (goal-oriented).
- Knowledge engineers use structured methods to increase the efficiency of the acquisition process.
- Knowledge Engineering is the process of eliciting Knowledge for any purpose be it Expert system or AI development

Views of knowledge engineering

There are two main views to knowledge engineering:

- Transfer View – This is the traditional view. In this view, the assumption is to apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligence systems.
- Modeling View – This is the alternative view. In this view, the knowledge engineer attempts to model the knowledge and problem solving techniques of the domain expert into the artificial intelligence system.

A major concern in knowledge engineering is the construction of ontologies. One philosophical question in this area is the debate between foundationalism and coherentism - are fundamental axioms of belief required, or merely consistency of beliefs which may have no lower-level beliefs to justify them?

Overview of Trends in Knowledge Engineering

Some of the trends in Knowledge Engineering in the last few years are discussed here.The text below is a brief overview of paper "Knowledge Engineering: Principles and methods" authored by Rudi Studer,V.Richard Benjamins and Dieter Fensel.

The paradigm Shift from a transfer view to a modeling view

According to the transfer view the human knowledge required to solve a problem is transferred and implemented into the knowledge base. However this assumes that concrete knowledge is already present in humans to solve a problem. The transfer view disregards the tacit knowledge an individual acquires in order to solve a problem. This is one of the reasons for a paradigm shift towards modeling view. This shift is compared to a shift from first generation expert systems to second generation expert systems.

The modeling view is a closer approximate of reality and perceives solving problems as a dynamic, cyclic, incessant process dependent on the knowledge acquired and the interpretations made by the system. This is similar to how an expert solves problems in real life.

The evolving of Role Limiting methods and Generic Tasks

Role limiting methods are based on reusable problem solving methods. Different knowledge roles are decided and the knowledge expected from each of these roles is clarified. However the disadvantage of role limiting methods is that there is no logical means of deciding whether a specific problem can be solved by a specific role-limiting method.

This disadvantage gave rise to Configurable role limiting methods. Configurable role limiting methods are based on the idea that a problem solving method can further be broken up into several smaller sub tasks each task solved by its own problem solving method.

Generic Tasks include a rigid knowledge structure, a standard strategy to solve problems, a specific input and a specific output.

The GT approach is based on the strong interaction problem hypothesis which states that the structure and representation of domain knowledge is completely determined by its use

The usage of Modeling Frameworks

The development of Specification languages and problem solving methods of knowledge based systems.Over the past few years the modeling frameworks that became prominent within Knowledge engineering are Common KADS, MIKE (Model-based and Incremental knowledge engineering) and PROTÉGÉ-II.PROTÉGÉ-II is a modeling framework influenced by the concept of ‘Ontology’.

The influence of Ontology

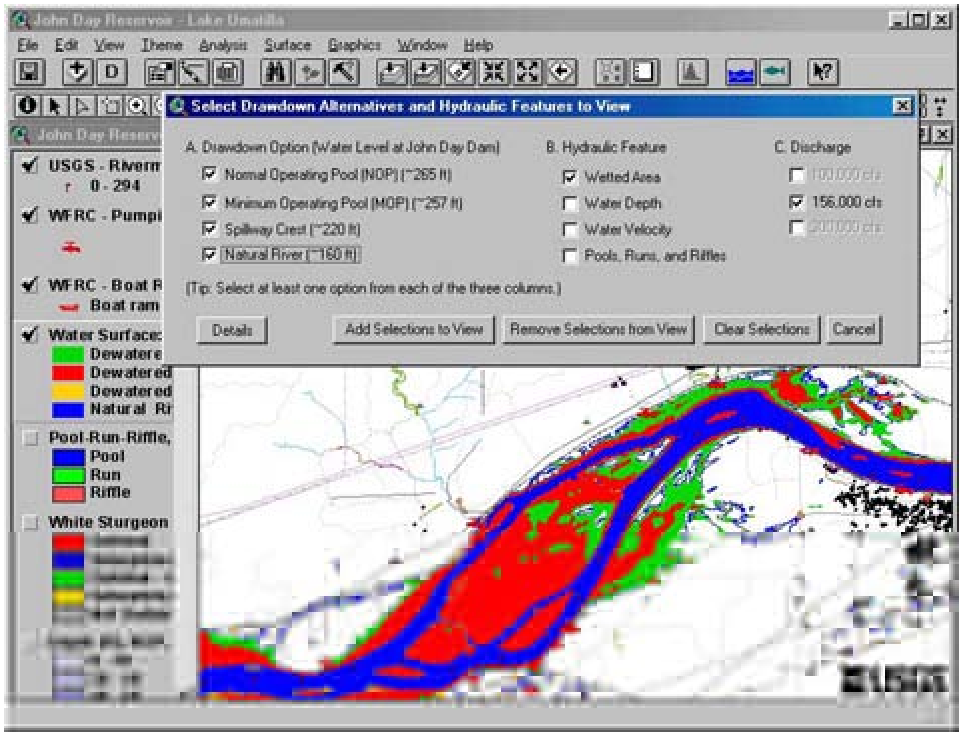
Ontologies help building model of a domain and define the terms inside the domain and the relationships between them. There are different types of Ontologies including Domain ontologies, Generic ontologies, application ontologies and representational ontologies.

While categorizing knowledge, storing, retrieving and managing information is not only useful for solving problems without direct need of human expertise but also leads to

‘Knowledge Management’ efforts that enable an organization to function efficiently in the long run.

Chapter-2

Decision Support System



Select Drawdown Alternatives and Hydraulic Features to View

A. Drawdown Option (Water Level at John Day Dam)
- [✓] Normal Operating Pool (NOP) (~265 ft)
- [✓] Minimum Operating Pool (MOP) (~257 ft)
- [✓] Spillway Crest (~220 ft)
- [✓] Natural River (~160 ft)

B. Hydraulic Feature
- [✓] Wetted Area
- [ ] Water Depth
- [ ] Water Velocity
- [ ] Pools, Runs, and Riffles

C. Discharge
- [ ] 100,000 cfs
- [✓] 156,000 cfs
- [ ] 200,000 cfs

(Tip: Select at least one option from each of the three columns.)

Example of a Decision Support System for John Day Reservoir.

A **decision support system** (**DSS**) is a computer-based information system that supports business or organizational decision-making activities. DSSs serve the management, operations, and planning levels of an organization and help to make decisions, which may be rapidly changing and not easily specified in advance.

DSSs include knowledge-based systems. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from a combination of raw data, documents, personal knowledge, or business models to identify and solve problems and make decisions.

Typical information that a decision support application might gather and present are:

- inventories of information assets (including legacy and relational data sources, cubes, data warehouses, and data marts),
- comparative sales figures between one period and the next,
- projected revenue figures based on product sales assumptions.

History

According to Keen (1978), the concept of decision support has evolved from two main areas of research: The theoretical studies of organizational decision making done at the Carnegie Institute of Technology during the late 1950s and early 1960s, and the technical work on interactive computer systems, mainly carried out at the Massachusetts Institute of Technology in the 1960s. It is considered that the concept of DSS became an area of research of its own in the middle of the 1970s, before gaining in intensity during the 1980s. In the middle and late 1980s, executive information systems (EIS), group decision support systems (GDSS), and organizational decision support systems (ODSS) evolved from the single user and model-oriented DSS.

According to Sol (1987) the definition and scope of DSS has been migrating over the years. In the 1970s DSS was described as "a computer based system to aid decision making". Late 1970s the DSS movement started focusing on "interactive computer-based systems which help decision-makers utilize data bases and models to solve ill-structured problems". In the 1980s DSS should provide systems "using suitable and available technology to improve effectiveness of managerial and professional activities", and end 1980s DSS faced a new challenge towards the design of intelligent workstations.

In 1987 Texas Instruments completed development of the Gate Assignment Display System (GADS) for United Airlines. This decision support system is credited with significantly reducing travel delays by aiding the management of ground operations at various airports, beginning with O'Hare International Airport in Chicago and Stapleton Airport in Denver Colorado.

Beginning in about 1990, data warehousing and on-line analytical processing (OLAP) began broadening the realm of DSS. As the turn of the millennium approached, new Web-based analytical applications were introduced.

The advent of better and better reporting technologies has seen DSS start to emerge as a critical component of management design. Examples of this can be seen in the intense amount of discussion of DSS in the education environment.

DSS also have a weak connection to the user interface paradigm of hypertext. Both the University of Vermont PROMIS system (for medical decision making) and the Carnegie

Mellon ZOG/KMS system (for military and business decision making) were decision support systems which also were major breakthroughs in user interface research. Furthermore, although hypertext researchers have generally been concerned with information overload, certain researchers, notably Douglas Engelbart, have been focused on decision makers in particular.

Taxonomies

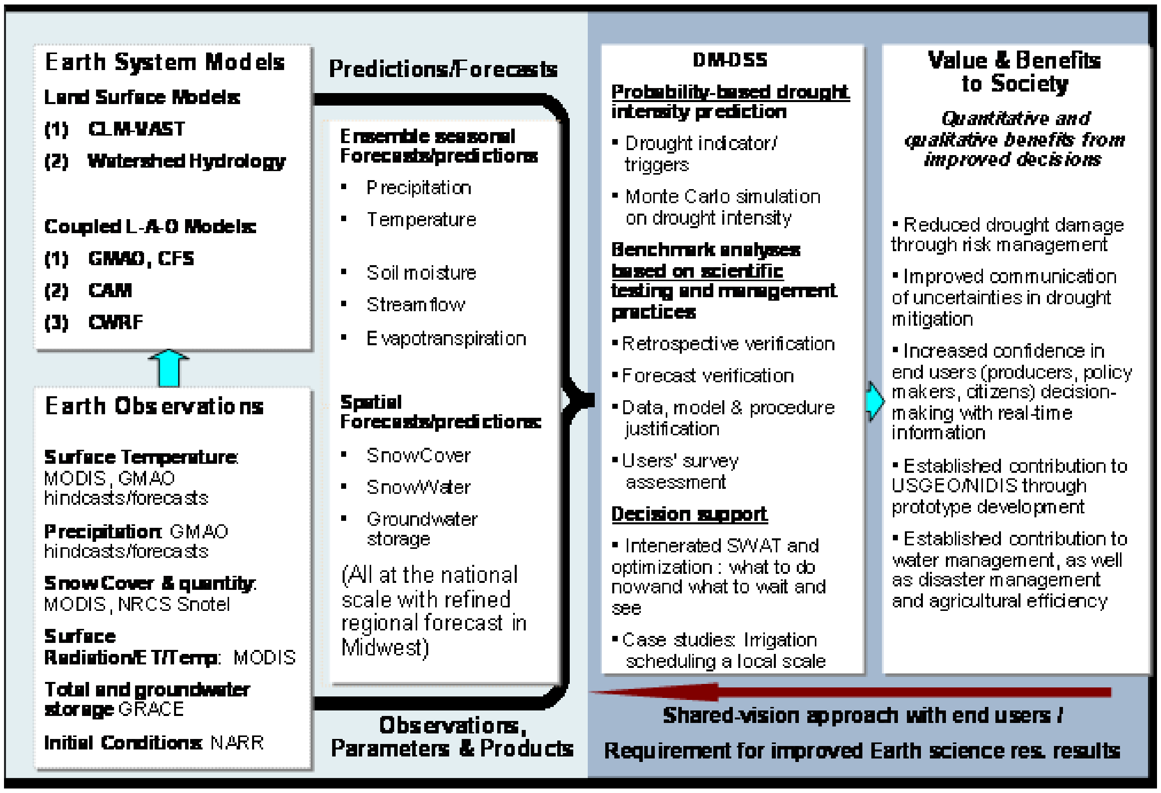
As with the definition, there is no universally-accepted taxonomy of DSS either. Different authors propose different classifications. Using the relationship with the user as the criterion, Haettenschwiler differentiates *passive*, *active*, and *cooperative DSS*. A *passive DSS* is a system that aids the process of decision making, but that cannot bring out explicit decision suggestions or solutions. An *active DSS* can bring out such decision suggestions or solutions. A *cooperative DSS* allows the decision maker (or its advisor) to modify, complete, or refine the decision suggestions provided by the system, before sending them back to the system for validation. The system again improves, completes, and refines the suggestions of the decision maker and sends them back to him for validation. The whole process then starts again, until a consolidated solution is generated.

Another taxonomy for DSS has been created by Daniel Power. Using the mode of assistance as the criterion, Power differentiates *communication-driven DSS*, *data-driven DSS*, *document-driven DSS*, *knowledge-driven DSS*, and *model-driven DSS*.

- A **communication-driven DSS** supports more than one person working on a shared task; examples include integrated tools like Microsoft's NetMeeting or Groove
- A **data-driven DSS** or data-oriented DSS emphasizes access to and manipulation of a time series of internal company data and, sometimes, external data.
- A **document-driven DSS** manages, retrieves, and manipulates unstructured information in a variety of electronic formats.
- A **knowledge-driven DSS** provides specialized problem-solving expertise stored as facts, rules, procedures, or in similar structures.
- A **model-driven DSS** emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model. Model-driven DSS use data and parameters provided by users to assist decision makers in analyzing a situation; they are not necessarily data-intensive. Dicodess is an example of an open source model-driven DSS generator.

Using scope as the criterion, Power differentiates *enterprise-wide DSS* and *desktop DSS*. An *enterprise-wide DSS* is linked to large data warehouses and serves many managers in the company. A *desktop, single-user DSS* is a small system that runs on an individual manager's PC.

Components



Earth System Models
Land Surface Models
(1) CLM-VAST
(2) Watershed Hydrology
Coupled L-A-O Models:
(1) GMAO, CFS
(2) CAM
(3) CWRF
Earth Observations
Surface Temperature: MODIS, GMAO hindcasts/forecasts
Precipitation: GMAO hindcasts/forecasts
Snow Cover & quantity: MODIS, NRCS Snotel
Surface Radiation/ET/Temp: MODIS
Total and groundwater storage: GRACE
Initial Conditions: NARR
Predictions/Forecasts
Ensemble seasonal Forecasts/predictions
Precipitation
Temperature
Soil moisture
Streamflow
Evapotranspiration
Spatial Forecasts/predictions:
SnowCover
SnowWater
Groundwater storage
(All at the national scale with refined regional forecast in Midwest)
Observations, Parameters & Products
DM-DSS
Probability-based drought intensity prediction
Drought indicator/ triggers
Monte Carlo simulation on drought intensity
Benchmark analyses based on scientific testing and management practices
Retrospective verification
Forecast verification
Data, model & procedure justification
Users' survey assessment
Decision support
Intenerated SWAT and optimization : what to do now and what to wait and see
Case studies: Irrigation scheduling a local scale
Value & Benefits to Society
Quantitative and qualitative benefits from improved decisions
Reduced drought damage through risk management
Improved communication of uncertainties in drought mitigation
Increased confidence in end users (producers, policy makers, citizens) decision-making with real-time information
Established contribution to USGEO/NIDIS through prototype development
Established contribution to water management, as well as disaster management and agricultural efficiency
Shared-vision approach with end users /
Requirement for improved Earth science res. results

Design of a Drought Mitigation Decision Support System.

Three fundamental components of a DSS architecture are:

1. the database (or knowledge base),
2. the model (i.e., the decision context and user criteria), and
3. the user interface.

The users themselves are also important components of the architecture.

Development Frameworks

DSS systems are not entirely different from other systems and require a structured approach. Such a framework includes people, technology, and the development approach.

DSS technology levels (of hardware and software) may include:

1. The actual application that will be used by the user. This is the part of the application that allows the decision maker to make decisions in a particular problem area. The user can act upon that particular problem.
2. Generator contains Hardware/software environment that allows people to easily develop specific DSS applications. This level makes use of case tools or systems such as Crystal, AIMMS, and iThink.

3. Tools include lower level hardware/software. DSS generators including special languages, function libraries and linking modules

An iterative developmental approach allows for the DSS to be changed and redesigned at various intervals. Once the system is designed, it will need to be tested and revised for the desired outcome.

Classification

There are several ways to classify DSS applications. Not every DSS fits neatly into one category, but may be a mix of two or more architectures.

Holsapple and Whinston classify DSS into the following six frameworks: Text-oriented DSS, Database-oriented DSS, Spreadsheet-oriented DSS, Solver-oriented DSS, Rule-oriented DSS, and Compound DSS.

A compound DSS is the most popular classification for a DSS. It is a hybrid system that includes two or more of the five basic structures described by Holsapple and Whinston.

The support given by DSS can be separated into three distinct, interrelated categories: Personal Support, Group Support, and Organizational Support.

DSS components may be classified as:

1. **Inputs:** Factors, numbers, and characteristics to analyze
2. **User Knowledge and Expertise:** Inputs requiring manual analysis by the user
3. **Outputs:** Transformed data from which DSS "decisions" are generated
4. **Decisions:** Results generated by the DSS based on user criteria

DSSs which perform selected cognitive decision-making functions and are based on artificial intelligence or intelligent agents technologies are called Intelligent Decision Support Systems (IDSS).

The nascent field of Decision engineering treats the decision itself as an engineered object, and applies engineering principles such as Design and Quality assurance to an explicit representation of the elements that make up a decision.

Applications

As mentioned above, there are theoretical possibilities of building such systems in any knowledge domain.

One example is the clinical decision support system for medical diagnosis. Other examples include a bank loan officer verifying the credit of a loan applicant or an engineering firm that has bids on several projects and wants to know if they can be competitive with their costs.

DSS is extensively used in business and management. Executive dashboard and other business performance software allow faster decision making, identification of negative trends, and better allocation of business resources.

A growing area of DSS application, concepts, principles, and techniques is in agricultural production, marketing for sustainable development. For example, the DSSAT4 package, developed through financial support of USAID during the 80's and 90's, has allowed rapid assessment of several agricultural production systems around the world to facilitate decision-making at the farm and policy levels. There are, however, many constraints to the successful adoption on DSS in agriculture.

DSS are also prevalent in forest management where the long planning time frame demands specific requirements. All aspects of Forest management, from log transportation, harvest scheduling to sustainability and ecosystem protection have been addressed by modern DSSs. A comprehensive list and discussion of all available systems in forest management is being compiled under the COST action Forsys

A specific example concerns the Canadian National Railway system, which tests its equipment on a regular basis using a decision support system. A problem faced by any railroad is worn-out or defective rails, which can result in hundreds of derailments per year. Under a DSS, CN managed to decrease the incidence of derailments at the same time other companies were experiencing an increase.

Benefits

1. Improves personal efficiency
2. Speed up the process of decision making
3. Increases organizational control
4. Encourages exploration and discovery on the part of the decision maker
5. Speeds up problem solving in an organization
6. Facilitates interpersonal communication
7. Promotes learning or training
8. Generates new evidence in support of a decision
9. Creates a competitive advantage over competition
10. Reveals new approaches to thinking about the problem space
11. Helps automate managerial processes

Chapter-3

NetWeaver Developer

NetWeaver Developer is a knowledgebase development system. This -

1. gives a brief history of the system,
2. summarizes key features of the software,
3. is a bit of a primer, describing basic attributes of a NetWeaver knowledgebase, and
4. provides secondary references that independently document some of the NetWeaver applications developed since the late 1980s.

First, though, a word about knowledgebases. While there are various ways of describing a knowledgebase, perhaps one of the more central concepts is that a knowledgebase provides a formal specification for interpreting information. Formal in this context means that the specification is ontologically committed to the semantics and syntax prescribed by a knowledgebase processor (aka, an engine).

A brief history

NetWeaver was created in late 1991 as a response to ease knowledge engineering tasks by giving a graphical user interface to the ICKEE (IConic Knowledge Engineering Environment) inference engine developed at Penn State University by Bruce J. Miller and Michael C. Saunders. The first iterations were simply a visual representation of dependency networks stored in a LISP-like syntax. NetWeaver quickly evolved into an interactive interface where the visual environment was also capable of editing the dependency networks and saving them in the ICKEE file format. Eventually NetWeaver became "live" in the sense that it could evaluate the dependency networks in real time.

NetWeaver basics

A NetWeaver knowledgebase graphically represents a problem to be evaluated as networks of topics, each of which evaluates a proposition. The formal specification of each topic is graphically constructed, and composed of other topics (e.g., premises) related by logic operators such as and, or, not, etc. NetWeaver topics and operators return a continuous-valued ‘‘truth value’’, that expresses the strength of evidence that the operator and its arguments provide to a topic or to another logic operator. The specification of an individual NetWeaver topic supports potentially complex reasoning because both topics and logic operators may be specified as arguments to an operator. Considered in its entirety, the complete knowledgebase specification for a problem can be thought of a mental map of the logical dependencies among propositions. In other words, the knowledgebase amounts to a formal logical argument in the classical sense.

When logic meets graphics

Cognitive theory suggests that human beings have two fundamental modes of reasoning: logical (albeit however informally some folks may do that when left to their own devices) and spatial. Interesting things happen when logic is implemented graphically.

First, the knowledge of individual subject-matter experts engaged in [[knowledge engineering]] often is not fully integrated when dealing with complex problems, at least initially. Rather, this knowledge may exist in a somewhat more loosely organized state, a sort of knowledge soup with chunks of knowledge floating about in it. A common observation of knowledge engineers experienced in graphically designing knowledgebases is that the process of constructing a graphic representation of problem-solving knowledge in a formal logical framework seems to be synergistic, with new insights into the expert’s knowledge emerging as the process unfolds.

Second, synergies similar to those observed in organizing the reasoning of individual subject-matter experts also can occur in knowledge engineering projects that require the interaction of multiple disciplines. For example, many different kinds of specialists may be involved in evaluating the overall health of a watershed. Use of a formal logic system, with well defined syntax and semantics, allows specialists’ representation of their problem solving approach to be expressed in a common language, which in turn facilitates understanding of how all the various perspectives of the different specialists fit together.

About NetWeaver knowledgebases

A NetWeaver knowledgebase has been defined by the developers as a network of networks (Miller and Saunders 2002). Each network corresponds to a topic of interest in the problem being evaluated by the knowledgebase.

NetWeaver knowledgebases are object-based. There are two basic types of objects: networks, and data links, each of which is represented in the logic structure by a programming object which has both state and behavior.

The NetWeaver engine is a Windows dynamic link library (DLL) developed by Rules of Thumb, Inc. (North East, PA). NetWeaver Developer is an interface to the engine that is used for designing knowledgebases.

Logic networks

A knowledgebase represents knowledge about how to solve a problem in terms of the topics of interest in the problem domain, and relations among these topics. Each logic network in a NetWeaver knowledge base represents a proposition about the condition of some ecosystem state or process.

- State - The key state variable of a logic network is its truth value which expresses the degree to which evidence from antecedent networks and data links support or refute the proposition. Logically, network A is said to be antecedent to network B if B depends upon A because network A must be evaluated before network B can be evaluated.
- Behavior - The basic function of a network is to evaluate the truth of its proposition. NetWeaver networks have three basic behaviors related to this function:
    - They query their antecedants to determine the latters' state.
    - They evaluate their own state, given the state of their antecedants.
    - They inform higher level networks that depend on them about their state.

Data links

A data link is an elementary dependency network with slightly modified behavior.

- State - Like a network, a data link may evaluate to a truth value, given a data input. A data link may also hold a data value that is subsequently transformed by mathematical operations defined for a calculated data link.
- Behavior:
    - In NetWeaver Developer, data links prompt the user for data input.
    - On receipt of data, data links evaluate their state, given the data input (simple data links), or pass the data value to a special data link that performs some transformation of input data (calculated data link).
    - They inform higher level networks that depend on them about their state.

Truth values

The truth value is the basic state variable of networks and data links. It expresses an observation's degree of membership in a set. Evaluations of degree of set membership are quantified in the semantics of fuzzy logic. Equivalently, think of the truth value metric as expressing the degree to which evidence supports the proposition of the network or data link; in EMDS, the symbology for maps displaying network truth values is based on the concept of strength of evidence.

Data links are frequently used to read a datum and evaluate its degree of membership in a concept that is quantified in a fuzzy argument (an argument that quantifies fuzzy set membership). Thus, in a data link the argument is a mathematical statement of a proposition. Some simple examples include:

- If the datum fully satisfies the argument, then the truth value of the data link is 1 (full support).
- If the datum is fully contrary to the argument, then the truth value of the data link is -1 (no support).
- If the datum partially satisfies the argument, then the truth value of the data link is in the open interval (-1, 1). Note in particular that negative truth values greater than -1 do not connote negative truth. Rather, such values connote low membership, or low support.
- If the data is not known, then the truth value of the data link is 0 (undetermined).

Interpretation of truth values within networks must be treated more generally, because the truth value of a network may depend on several to many logic operators. Simple examples related to the two key logic operators, AND and OR, are:

- If ‘’’all’’’ logical antecedents to an AND operator fully support the AND relation, then the truth value of the operator is 1 (full support).
- If ‘’’any’’’ logical antecedent to an AND operator is fully contrary to the AND

relation, then the truth value of the operator is -1 (no support).

- If ‘’’any’’’ logical antecedent to an OR operator fully supports the OR relation, then the truth value is 1 (full support).
- If there is no evidence for or against an AND or OR relation, then the truth value of either operator is 0 (undetermined).

As with data links, networks may also evaluate to partially true. Two conditions give rise to this condition in NetWeaver:

- One or more data items are missing and cannot be supplied, and therefore contribute a value of 0 to an AND.
- One or more data items that influence the truth value of a dependency network have been evaluated against a fuzzy argument and found not to have full membership in the fuzzy set defined by the fuzzy argument (the data provides only partial support for the proposition).

Chapter-4

Ontology (Information Science)

In computer science and information science, an **ontology** is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. It is used to reason about the entities within that domain, and may be used to describe the domain. Its meaning is vastly different from the word Ontology in philosophy.

In theory, an ontology is a "formal, explicit specification of a shared conceptualisation". An ontology provides a shared vocabulary, which can be used to model a domain — that is, the type of objects and/or concepts that exist, and their properties and relations.

Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it. The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework.

Overview

The term *ontology* has its origin in philosophy, and has been applied in many different ways. The word "ontology" comes from the Greek *ὄν* (on), which literally means 'existence'. The core meaning within computer science is a model for describing the world that consists of a set of types, properties, and relationship types. Exactly what is provided around these varies, but they are the essentials of an ontology. There is also generally an expectation that there be a close resemblance between the real world and the features of the model in an ontology.

What many ontologies have in common in both computer science and in philosophy is the representation of entities, ideas, and events, along with their properties and relations, according to a system of categories. In both fields, one finds considerable work on problems of ontological relativity (e.g., Quine and Kripke in philosophy, Sowa and Guarino in computer science), and debates concerning whether a normative ontology is viable (e.g., debates over foundationalism in philosophy, debates over the Cyc project in AI). Differences between the two are largely matters of focus. Philosophers are less concerned with establishing fixed, controlled vocabularies than are researchers in computer science, while computer scientists are less involved in discussions of first principles (such as debating whether there are such things as fixed essences, or whether entities must be ontologically more primary than processes).

History

Historically, ontologies arise out of the branch of philosophy known as metaphysics, which deals with the nature of reality – of what exists. This fundamental branch is concerned with analyzing various types or modes of existence, often with special attention to the relations between particulars and universals, between intrinsic and extrinsic properties, and between essence and existence. The traditional goal of ontological inquiry in particular is to divide the world "at its joints", to discover those fundamental categories, or kinds, into which the world's objects naturally fall.

During the second half of the 20th century, philosophers extensively debated the possible methods or approaches to building ontologies, without actually *building* any very elaborate ontologies themselves. By contrast, computer scientists were building some large and robust ontologies (such as WordNet and Cyc) with comparatively little debate over *how* they were built.

Since the mid-1970s, researchers in the field of artificial intelligence (AI) have recognized that capturing knowledge is the key to building large and powerful AI systems. AI researchers argued that they could create new ontologies as computational models that enable certain kinds of automated reasoning. In the 1980s, the AI community began to use the term *ontology* to refer to both a theory of a modeled world and a component of knowledge systems. Some researchers, drawing inspiration from philosophical ontologies, viewed computational ontology as a kind of applied philosophy.

In the early 1990s, the widely cited Web page and paper "Toward Principles for the Design of Ontologies Used for Knowledge Sharing" by Tom Gruber is credited with a deliberate definition of *ontology* as a technical term in computer science. Gruber "introduced the term to mean a specification of a conceptualization. That is "an ontology is a description (like a formal specification of a program) of the concepts and relationships that can formally exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set of concept definitions, but more general. And it is a different sense of the word than its use in philosophy".

According to Gruber (1993) "ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions – that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world. To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms.

In the early years of the 21st century, the interdisciplinary project of cognitive science has been bringing the two circles of scholars closer together. For example, there is talk of a "computational turn in philosophy" that includes philosophers analyzing the formal ontologies of computer science (sometimes even working directly with the software), while researchers in computer science have been making more references to those philosophers who work on ontology (sometimes with direct consequences for their methods). Still, many scholars in both fields are uninvolved in this trend of cognitive science, and continue to work independently of one another, pursuing separately their different concerns.

Ontology components

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals (instances), classes (concepts), attributes, and relations. Here each of these components is discussed in turn.

Common components of ontologies include:

- Individuals: instances or objects (the basic or "ground level" objects)
- Classes: sets, collections, concepts, classes in programming, types of objects, or kinds of things
- Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have
- Relations: ways in which classes and individuals can be related to one another
- Function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement
- Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input
- Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form
- Axioms: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axioms" in generative grammar and formal logic. In those disciplines, axioms include only statements asserted as *a priori* knowledge. As used here, "axioms" also include the theory derived from axiomatic statements
- Events: the changing of attributes or relations

Ontologies are commonly encoded using ontology languages.

Domain ontologies and upper ontologies

A domain ontology (or domain-specific ontology) models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. For example the word *card* has many different meanings. An ontology about the domain of poker would model the "playing card" meaning of the word, while an ontology about the domain of computer hardware would model the "punched card" and "video card" meanings.

An upper ontology (or foundation ontology) is a model of the common objects that are generally applicable across a wide range of domain ontologies. It employs a core glossary that contains, the terms, and associated object descriptions, as they are used in various, relevant domain sets. There are several standardized upper ontologies available for use, including Dublin Core, GFO, OpenCyc/ResearchCyc, SUMO, and DOLCE. WordNet, while considered an upper ontology by some, is not strictly an ontology. However, it has been employed as a linguistic tool for learning domain ontologies.

The Gellish ontology is an example of a combination of an upper and a domain ontology.

Since domain ontologies represent concepts in very specific and often eclectic ways, they are often incompatible. As systems that rely on domain ontologies expand, they often need to merge domain ontologies into a more general representation. This presents a challenge to the ontology designer. Different ontologies in the same domain can also arise due to different perceptions of the domain based on cultural background, education, ideology, or because a different representation language was chosen.

At present, merging ontologies that are not developed from a common foundation ontology is a largely manual process and therefore time-consuming and expensive. Domain ontologies that use the same foundation ontology to provide a set of basic elements with which to specify the meanings of the domain ontology elements can be merged automatically. There are studies on generalized techniques for merging ontologies, but this area of research is still largely theoretical.

Ontology engineering

Ontology engineering (or ontology building) is a subfield of knowledge engineering that studies the methods and methodologies for building ontologies. It studies the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

Ontology engineering aims to make explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. Ontology engineering offers a direction towards solving the interoperability problems brought about by semantic obstacles, such as the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain.

Ontology languages

An ontology language is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- Common Algebraic Specification Language is a general logic-based specification language developed within the IFIP working group 1.3 "Foundations of System Specifications" and functions as a de facto standard in the area of software specifications. It is now being applied to ontology specifications in order to provide modularity and structuring mechanisms.
- Common logic is ISO standard 24707, a specification for a family of ontology languages that can be accurately translated into each other.
- The Cyc project has its own ontology language called CycL, based on first-order predicate calculus with some higher-order extensions.
- DOGMA (Developing Ontology-Grounded Methods and Applications) adopts the fact-oriented modeling approach to provide a higher level of semantic stability.
- The Gellish language includes rules for its own extension and thus integrates an ontology with an ontology language.
- IDEF5 is a software engineering method to develop and maintain usable, accurate, domain ontologies.
- KIF is a syntax for first-order logic that is based on S-expressions.
- Rule Interchange Format (RIF) and F-Logic combine ontologies and rules.
- OWL is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.
- Semantic Application Design Language (SADL) captures a subset of the expressiveness of OWL, using an English-like language entered via an Eclipse Plug-in.
- OBO, a language used for biological and biomedical ontologies.
- (E)MOF and UML are standards of the OMG

Examples of published ontologies

- Basic Formal Ontology, a formal upper ontology designed to support scientific research
- BioPAX, an ontology for the exchange and interoperability of biological pathway (cellular processes) data
- BMO, an e-Business Model Ontology based on a review of enterprise ontologies and business model literature
- CCO (Cell Cycle Ontology), an application ontology that represents the cell cycle
- CContology (Customer Complaint Ontology), an e-business ontology to support online customer complaint management
- CIDOC Conceptual Reference Model, an ontology for cultural heritage

- COSMO, a Foundation Ontology (current version in OWL) that is designed to contain representations of all of the primitive concepts needed to logically specify the meanings of any domain entity. It is intended to serve as a basic ontology that can be used to translate among the representations in other ontologies or databases. It started as a merger of the basic elements of the OpenCyc and SUMO ontologies, and has been supplemented with other ontology elements (types, relations) so as to include representations of all of the words in the Longman dictionary defining vocabulary.
- Cyc, a large Foundation Ontology for formal representation of the universe of discourse.
- Disease Ontology, designed to facilitate the mapping of diseases and associated conditions to particular medical codes
- DOLCE, a Descriptive Ontology for Linguistic and Cognitive Engineering
- Dublin Core, a simple ontology for documents and publishing
- Foundational, Core and Linguistic Ontologies
- Foundational Model of Anatomy, an ontology for human anatomy
- Friend of a Friend, an ontology for describing persons, their activities and their relations to other people and objects
- Gene Ontology for genomics
- Gellish English dictionary, an ontology that includes a dictionary and taxonomy that includes an upper ontology and a lower ontology that focusses on industrial and business applications in engineering, technology and procurement.
- Geopolitical ontology, an ontology describing geopolitical information created by Food and Agriculture Organization(FAO). The geopolitical ontology includes names in multiple languages (English, French, Spanish, Arabic, Chinese, Russian and Italian); maps standard coding systems (UN, ISO, FAOSTAT, AGROVOC, etc); provides relations among territories (land borders, group membership, etc); and tracks historical changes. In addition, FAO provides web services of geopolitical ontology and a module maker to download modules of the geopolitical ontology into different formats (RDF, XML, and EXCEL).
- GOLD, General Ontology for Linguistic Description
- GUM (Generalized Upper Model), a linguistically-motivated ontology for mediating between clients systems and natural language technology
- IDEAS Group, a formal ontology for enterprise architecture being developed by the Australian, Canadian, UK and U.S. Defence Depts.
- Linkbase, a formal representation of the biomedical domain, founded upon Basic Formal Ontology.
- LPL, Lawson Pattern Language
- NIFSTD Ontologies from the Neuroscience Information Framework: a modular set of ontologies for the neuroscience domain**.**
- OBO Foundry, a suite of interoperable reference ontologies in biomedicine
- Ontology for Biomedical Investigations, an open access, integrated ontology for the description of biological and clinical investigations
- OMNIBUS Ontology, an ontology of learning, instruction, and instructional design
- Plant Ontology for plant structures and growth/development stages, etc.

- POPE, Purdue Ontology for Pharmaceutical Engineering
- PRO, the Protein Ontology of the Protein Information Resource, Georgetown University.
- Program abstraction taxonomy program abstraction taxonomy
- Protein Ontology for proteomics
- Suggested Upper Merged Ontology, a formal upper ontology
- Systems Biology Ontology (SBO), for computational models in biology
- SWEET, Semantic Web for Earth and Environmental Terminology
- ThoughtTreasure ontology
- TIME-ITEM, Topics for Indexing Medical Education
- UMBEL, a lightweight reference structure of 20,000 subject concept classes and their relationships derived from OpenCyc
- WordNet, a lexical reference system
- YAMATO, Yet Another More Advanced Top-level Ontology

The W3C Linking Open Data Community Project coordinates attempts to converge different ontologies into worldwide Data Web.

Ontology libraries

The development of ontologies for the Web has led to the emergence of services providing lists or directories of ontologies with search facility. Such directories have been called ontology libraries.

The following are static libraries of human-selected ontologies.

- DAML Ontology Library maintains a legacy of ontologies in DAML.
- Protege Ontology Library contains a set of OWL, Frame-based and other format ontologies.
- SchemaWeb is a directory of RDF schemata expressed in RDFS, OWL and DAML+OIL.

The following are both directories and search engines. They include crawlers searching the Web for well-formed ontologies.

- OBO Foundry / Bioportal is a suite of interoperable reference ontologies in biology and biomedicine.
- OntoSelect Ontology Library offers similar services for RDF/S, DAML and OWL ontologies.
- Ontaria is a "searchable and browsable directory of semantic web data", with a focus on RDF vocabularies with OWL ontologies. (NB Project "on hold" since 2004).
- Swoogle is a directory and search engine for all RDF resources available on the Web, including ontologies.

Chapter-5

Knowledge Modeling and Ontology Engineering

Knowledge modeling

Knowledge modeling is a process of creating a computer interpretable model of knowledge or standard specifications about a kind of process and/or about a kind of facility or product. The resulting knowledge model can only be computer interpretable when it is expressed in some knowledge representation language or data structure that enables the knowledge to be interpreted by software and to be stored in a database or data exchange file.
Knowledge-based engineering or knowledge-aided design is a process of computer-aided usage of such knowledge models for the design of products, facilities or processes. The design of products or facilities then uses the knowledge model to guide the creation of the facility or product that need to be designed. In other words it used knowledge about a kind of object to create a product model of an (imaginary) individual object. Similarly, the design of a particular process implies the creation of a process model, which design activity can be guided by the knowledge that is contained in a knowledge model about such a kind of process. The resulting process model, product model or facility model is typically also stored in a database.

Usually the knowledge representation language only allows to represent knowledge (about kinds of things), whereas another language or data structure is required to represent and store the information models about individual things. If the knowledge representation language enables to express both, then the knowledge model and the information model can be expressed in the same language (or data structure). An example of a language that enables the expression of knowledge as well as information about individual things is Gellish English.

The basis of a knowledge model of an assembly physical object is a decomposition structure that specifies the components of the assembly and possible the sub-components of the components. For example, knowledge about a compressor system includes that a compressor system consists of a compressor, a lubrication system, etc, whereas a lubrication system consists of a pump system, etc. Assume that this knowledge is expressed in a knowledge representation language that expresses knowledge as a collection of relations between two kinds of things, whereas in that language a relation type is defined that is called <shall have as part a>. Then a part of a knowledge model about a compressor system will consist of the following expressions of knowledge facts:

- compressor system shall have as part a compressor
- compressor system shall have as part a lubrication system
- lubrication system shall have as part a pump system
- pump system shall have as part a pump

Such a knowledge model will be further extended with knowledge and specifications about the properties of the components, their fabrications and possibly testing and maintenance requirements.

Similarly, a knowledge model of a process is basically a specification of the sequence of process stages. This sequence is determined by the fact that a kind of stream is output of a kind of process stage, whereas that same type of stream in input in the next process stage. So the defined streams have roles as inputs to process stages, whereas the same streams are outputs of other process stages. For example:

- water shall be input in a boiler
- steam shall be output of a boiler
- steam shall be input in a heater
- condensate shall be output of a heater
- etc.

Explicitation of document content

Knowledge modeling includes the explicitation of knowledge and requirements that is available in documents, such as design manuals, (international) standard specifications and standard data sheets. In order to make such knowledge computer interpretable it need to be expressed in a formal knowledge representation language and thus transformed into a computer interpretable form. For example in the form of an expressions Gellish English. This enables that the knowledge and requirements are related to the objects in the knowledge model, whereas the whole model is again stored in a Database.
The knowledge that is contained in documents can be modeled at various levels of explicitation. A low level of explicitation keeps large parts of the specifications in the form of natural language text. This means that the text is only human interpretable, but is nevertheless related to the objects in the knowledge model. Thus software can still present the information to users when knowledge about that object is requested. The other extreme is that the content of each sentence in a documents is converted in the formal knowledge representation language and thus the objects that are mentioned in those sentences become an integral part of the computer interpretable knowledge model. For

example, the knowledge that the API 617 standard contains a standard specification for compressors can be linked to the concept compressor in the knowledge model of a compressor system. This can be expressed in a knowledge representation language (using the relation type <is specified in> as follows:

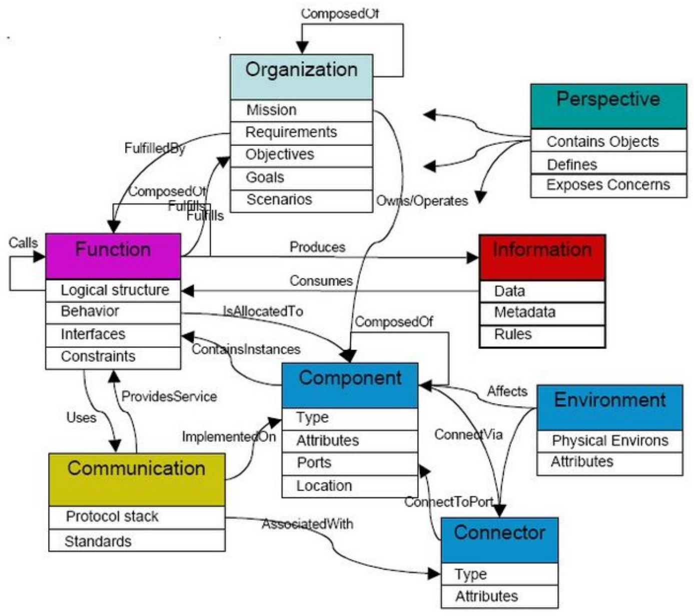
- compressor <is specified in> API 617

A higher level of explicitation means that paragraphs or sentences in natural language are related to components in the knowledge model. A full explicit model means that the natural language sentences are completely transformed into data in a database structure. For example, a specification of a minimum shaft diameter might be included in the knowledge model as follows:

- shaft diameter <shall have on scale a value greater than> 20 mm

The above described explicitation process results in Knowledge Models and Standard Specifications Models that enable their use for computer supported knowledge-aided design as well as for automated verification of designs.

Ontology engineering



Example of a constructed MBED Top Level Ontology based on the Nominal set of views.

Ontology engineering in computer science and information science is a new field, which studies the methods and methodologies for building ontologies: formal representations of a set of concepts within a domain and the relationships between those concepts. A large scale representation of abstract concepts such as actions, time, physical objects and beliefs would be an example of ontological engineering.

Overview

[Ontology engineering] aims at making explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. Ontology engineering offers a direction towards solving the inter-operability problems brought about by semantic obstacles, i.e. the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain.

Ontologies provide a common vocabulary of an area and define, with different levels of formality, the meaning of the terms and the relationships between them. During the last decade, increasing attention has been focused on ontologies. Ontologies are now widely used in knowledge engineering, artificial intelligence and computer science; in applications related to areas such as knowledge management, natural language processing, e-commerce, intelligent information integration, bio-informatics, education; and in new emerging fields like the semantic web. Ontological engineering is a new field of study concerning the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

Ontology languages

An ontology language is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- Common logic is ISO standard 24707, a specification for a family of ontology languages that can be accurately translated into each other.
- The Cyc project has its own ontology language called CycL, based on first-order predicate calculus with some higher-order extensions.
- The Gellish language includes rules for its own extension and thus integrates an ontology with an ontology language.
- IDEF5 is a software engineering method to develop and maintain usable, accurate, domain ontologies.
- KIF is a syntax for first-order logic that is based on S-expressions.
- Rule Interchange Format (RIF) and F-Logic combine ontologies and rules.
- OWL is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.
- XBRL (Extensible Business Reporting Language) is a syntax for expressing business semantics.

Ontology Engineering In Life Sciences

Life sciences is flourishing with ontologies that biologists use to make sense of their experiments. For inferring correct conclusions from experiments, ontologies have to be structured optimally against the knowledge base they represent. The structure of an ontology needs to be changed continuously so that it is an accurate representation of the underlying domain.

Recently, an automated method was introduced for engineering ontologies in life sciences such as Gene Ontology (GO), one of the most successful and widely used biomedical ontology. Based on information theory, it restructures ontologies so that the levels represent the desired specificity of the concepts. Similar information theoretic approaches have also been used for optimal partition of Gene Ontology. Given the mathematical nature of such engineering algorithms, these optimizations can be automated to produce a principled and scalable architecture to restructure ontologies such as GO.

Tools for ontology engineering

- DOGMA
- DogmaModeler
- KAON
- OntoClean
- OnToContent
- HOZO
- Protégé (software)

Chapter-6

Knowledge-based Engineering

Knowledge-based engineering (KBE) is a discipline with roots in computer-aided design (CAD) and knowledge-based systems but has several definitions and roles depending upon the context. An early role was support tool for a design engineer generally within the context of product design. Success of early KBE prototypes was remarkable; eventually this led to KBE being considered as the basis for generative design with many expectations for hands-off performance where there would be limited human involvement in the design process.

Overview

KBE can be defined as engineering on the basis of electronic knowledge models. Such knowledge models are the result of knowledge modeling that uses knowledge representation techniques to create the computer interpretable models. The knowledge models can be imported in and/or stored in specific engineering applications that enable engineers to specify requirements or create designs on the basis of the knowledge in such models. There are various methods available for the development of knowledge models, most of them are system dependent. An example of a system-independent language for the development machine-readable ontology databases, including support for basic engineering knowledge, is called Gellish English. An example of a CAD-specific system that can store knowledge and use it for design is the CATIA program through its KnowledgeWare module. An example of a CAD-independent, language-based KBE system with full compiler and support for runtime application deployment is General-purpose Declarative Language (GDL) from Genworks.

KBE can have a wide scope that covers the full range of activities related to Product Lifecycle Management and Multidisciplinary design optimization. KBE's scope would include design, analysis (computer-aided engineering – CAE), manufacturing, and support. In this inclusive role, KBE has to cover a large multi-disciplinary role related to many computer aided technologies (CAx).

KBE also has more general overtones. One of its roles is to bridge knowledge management and design automation. Knowledge processing is a recent advance in computing. It has played a successful role in engineering and is now undergoing modifications (to be explained). An example of KBE's role is generative mechanical design. There are others. KBE can be thought of as an advanced form of computer applications (in some forms with an extreme end-user computing flavor) that support PLM and CAx.

There are similar techniques, such as electronic design automation. AAAI provides a long list of engineering applications. some of which are within the KBE umbrella. At some point, the concept of KBE might split into several sub-categories as MCAD and ECAD are just two of many possible types of design automation.

History

KBE essentially was a complementary development to CAx and can be dated from the 1980s. CAx has been developing along with the computer after making large strides in the 1970s.

As with any bit of progress, KBE flashed on the horizon, lit the sky for a while, and then experienced a downslide. KBE had sufficient success stories that sustained it long enough into the 1990s to get attention. Some prime contributors to the hiatus of KBE were unmanageable expectations, increasing tedium associated with forming completion of results, and some notion that the architecture for KBE was not sufficiently based upon the newer technology.

KBE continued to exist in pockets. With the prevalence of object-oriented methods, systems advanced enough to allow re-implementation. This reconstruction has been on-going for several years and has been frustratingly slow. Now, with the basis for this discipline becoming more robust, it starts to get interesting again.

KBE, as implemented with ICAD can be thought of as an advanced form of computer applications (in some forms with an extreme end-user computing flavor) that support PLM and CAx.

KBE and product lifecycle management

The scope of PLM involves all the steps that exist within any industry that produces goods. KBE at this level will deal with product issues of a more generic nature than it will with CAx. Some might call this level 'assembly' in orientation. However, it's much more than that as PLM covers both the technical and the business side of a product.

KBE then needs to support the decision processes involved with configuration, trades, control, management, and a number of other areas, such as optimization.

Recently the Object Management Group released a RFP document and requested feedback.

KBE and CAX

CAx crosses many disciplinary bounds and provides a sound basis for PLM. In a sense, CAx is a form of applied science that uses most of the disciplines of engineering and their associated fields. Materials science comes to mind.

KBE's support of CAx may have some similarities with its support of PLM but, in a sense, the differences are going to be larger.

The KBE flavor at the CAx level may assume a strong behavioral flavor. Given the underlying object oriented focus, there is a natural use of entities possessing complicated attributes and fulfilling non-trivial roles. One vendor's approach provides a means via workbenches to embed attributes and methods within sub-parts (object) or within a joining of sub-parts into a part.

As an aggregate, the individual actions, that are event driven, can be fairly involved. This fact identifies one major problem, namely control of what is essentially a non-deterministic mixture. This characteristic of the decision problem will get more attention as the KBE systems subsume more levels and encompasses a broader scope of PLM.

KBE and knowledge management

KBE is related to knowledge management which has many levels itself. Some approaches to knowledge are reductionistic, as well they ought to be given the pragmatic focus of knowledge modeling. However, due to KBE dealing with aggregates that can be quite complicated both in structure and in behavior, some holistic notions (note link to complexity theory) might be apropos.

Also, given all the layers of KBE and given the fact that one part of an associated space is heavily mathematical (namely, manifold in nature), KBE is extremely interesting from the knowledge viewpoint (or one would hope).

All one has to do is note that the KBE process's goal is to produce results in the 'real world' via artifacts and to do so using techniques that are highly computational. That, in essence, is the epitome of applied science/engineering, and it could never be non-interesting.

KBE methodology

The development of KBE applications concerns the requirements to identify, capture, structure, formalize and finally implement knowledge. Many different so-called KBE platforms support only the implementation step which is not always the main bottleneck in the KBE development process. In order to limit the risk associated with the development and maintenance of KBE application there is a need to rely on an appropriate methodology for managing the knowledge and maintaining it up to date. As example of such KBE methodology the EU project MOKA "Methodology and tools Oriented to Knowledge based Applications" propose solutions which focus on the structuration and formalization steps as well as links to the implementation.

An alternative to MOKA is to use a general methodology for developing knowledge bases for expert systems and for intranet pages. Such a methodology is described in "Knowledge Acquisition in Practice: A Step-by-step Guide" by Nick Milton.

Languages for KBE

Some questions can be asked in regard to KBE implementation: can we represent knowledge in a vendor-neutral format? can the knowledge in our designs be retained for decades, long after a vendor system (such as CATIA) has disappeared?

These questions are addressed in a 2005 Aerospace COE presentation A Proposal for CATIA V6 by Walter Wilson of Lockheed Martin.

Mr. Wilson advocates using a type of programming language to define design data—operations, parameters, formulas, etc. -- instead of a proprietary file format (such as Dassault's CATIA). One's data would no longer be tied to a specific CAD system. Unlike STEP, which inevitably lags commercial CAD systems in the features it supports, programmability would allow the definition of new design features.

A logic programming language is proposed as the basis for the engineering design language because of its simplicity and extensibility. The geometric engine for the language features would be open source to give engineers control over approximation algorithms and to better guarantee long-term accessibility of the data.

Meanwhile, the commercially available General-purpose Declarative Language (GDL) from Genworks International addresses the issue of application longevity by providing a high-level declarative language kernel which is a superset of a standard dialect of the Lisp programming language (currently ANSI Common Lisp, or CL).

The GDL kernel follows a concise, pragmatic language specification representing something akin to a *de-facto* neutral format for representing KBE-style knowledge. It consists of the same Smalltalk-inspired declarative object-oriented (and object-centric) message-passing format which been a common thread among classical KBE systems for more than two decades. While CL is a multi-paradigm language (supporting procedural, object-oriented, and functional programming), the core features of KBE tend to share several aspects with Functional programming languages "under the hood" (e.g. lazy evaluation, immutable data). However there is a consensus that pure Functional programming is too esoteric for typical engineers to practice, so one of the purposes of a declarative KBE language such as GDL is to provide an "engineer-friendly" object-centric front-end on what is essentially a Functional language programming environment.

Because GDL applications are written as a strict superset of a standard Lisp dialect, only the high-level declarative surface syntax is GDL-specific. The bulk of application code is purely compliant with the underlying language standard. And because of Lisp's inherent (and unique) support for code transformation macros, even this surface syntax is subject to straightforward automated conversion among other variations of the de-facto standard. It is reasonable to expect that implementations following this approach will eventually converge on a true vendor-neutral Standard KBE language specification.

The Pacelab Suite addresses the problem that functional programming languages are still not widely accepted by potential KBE users. Engineers are usually trained in procedural and object-oriented programming languages; in quite a few software environments (Excel, MATLAB and FORTRAN) used by engineers the procedural programming style clearly dominates. Therefore the Pacelab Suite rebases the inherent technological advantages offered by a development and runtime environment like Common Lisp, on a new technological paradigm (.NET Framework) equally supporting procedural, object-oriented and functional languages.

KBE in Academia

- Knowledge-based engineering at the Norwegian University of Science and Technology (NTNU)
- Knowledge Based Engineering department at the Faculty of Aerospace Engineering of the Delft University of Technology

Implementations

The following KBE development packages are commercially available:

For CAD

- Adaptive Modeling Language from TechnoSoft Inc.
- DriveWorks A SolidWorks Certified Gold Partner
- GDL from Genworks International
- Kadviser from NIMTOTH previously edited by Kade-Tech
- KBEWorks by VisionKBE
- Knowledge Fusion from Siemens PLM Software
- Knowledgeware from Dassault Systemes
- Magix by Navitech
- Pro/ENGINEER Expert Framework from Parametric Technology Corporation
- SmartAssembly for Pro/ENGINEER from Sigmaxim Inc
- TactonWorks Interactive design automation inside SolidWorks
- YVE - Your Variant Engineer from tecneos software-engineering
- ICAD from Dassault Systemes (no longer available)
- KBMax Configurator by Citius Corporation

For General-purpose development of Web-deployed applications

- GDL from Genworks International

For analysis, design and engineering processes

- GDL from Genworks International
- Pacelab Suite by PACE Aerospace Engineering and Information Technology GmbH
- PCPACK by Tacit Connexions
- Quaestor by Qnowledge Modeling Technologies

KBE futures, KBE theory

KBE, as a particular example of KS, is a multi-disciplinary framework that has more than practical considerations. Not only will KBE require successful handling of issues of the computational (Ontology, Artificial Intelligence, Entscheidungsproblem, Interactive computation, Category Theory, ...) and logic (non-monotonic issues related to the qualification, frame, and ramification problems)), it will touch upon all sciences that deal with matter, its manipulations, and the related decisions. In a sense, PLM allows us to have the world as a large laboratory for experimental co-evolution of our knowledge and our artificial co-horts. As noted in the ACM Communications, "Computers will grow to become scientists in their own right, with intuitions and computational variants of fascination and curiosity." What better framework is there to explore the "increasingly complicated mappings between the human world and the computational"?

In terms of methodology and their associated means, KBE offers support via several paradigms. These range from the home-grown all the way to strategically defined and integrated tools that cover both breadth and depth. A continuing theme will be resolving the contextual definitions for KBE into a coherent discipline (or at least attempting this) and keeping a handle on managing the necessary quantitative comparisons. One issue of importance considers what limits there may be to the computational; this study requires a multi-disciplinary focus and an understanding of the quasi-empirical. Given the knowledge focus of KBE, another issue involves what limits there might be to a computational basis for knowledge and whether these are overcome with the more advanced types of human-machine interface.

Chapter-7

Knowledge Representation and Reasoning

Knowledge representation (KR) and reasoning' is an area of artificial intelligence whose fundamental goal is to represent knowledge in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge. It analyzes how to formally think - how to use a symbol system to represent a domain of discourse (that which can be talked about), along with functions that allow inference (formalized reasoning) about the objects. Generally speaking, some kind of logic is used both to supply formal semantics of how reasoning functions apply to symbols in the domain of discourse, as well as to how to supply operators such as quantifiers, modal operators, etc. that, along with an interpretation theory, give meaning to the sentences in the logic.

When we design a knowledge representation (and a knowledge representation system to interpret sentences in the logic in order to derive inferences from them) we have to make choices across a number of design spaces. The single most important decision to be made, is the *expressivity* of the KR. The more expressive, the easier and more compact it is to "say something". However, more expressive languages are harder to automatically derive inferences from. An example of a less expressive KR would be propositional logic. An example of a more expressive KR would be autoepistemic temporal modal logic. Less expressive KRs may be both complete and consistent (formally less expressive than set theory). More expressive KRs may be neither complete nor consistent.

The key problem is to find a KR and a supporting reasoning system that can make the inferences your application needs within the resource constraints appropriate to the problem at hand. Recent developments in KR have been driven by the Semantic Web, and have included development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

Overview

In field there are a number of representation techniques such as frames, rules, tagging, and semantic networks which have originated from theories of human information processing. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to represent knowledge in a manner which will facilitate reasoning (aka inferencing or drawing conclusions); knowledge representation and reasoning being seen as two sides of a coin. A good knowledge representation must be both declarative and procedural knowledge. What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand :

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world."

Some issues that arise in knowledge representation from an AI perspective are:

- How do people represent knowledge?
- What is the nature of knowledge?
- Should a representation scheme deal with a particular domain or should it be general purpose?
- How expressive is a representation scheme or formal language?
- Should the scheme be declarative or procedural?

There has been very little top-down discussion of the knowledge representation (KR) issues and research in this area is a well aged quillwork. There are well known problems such as "spreading activation" (this is a problem in navigating a network of nodes), "subsumption" (this is concerned with selective inheritance; e.g. an ATV can be thought of as a specialization of a car but it inherits only particular characteristics) and "classification." For example a tomato could be classified both as a fruit and a vegetable.

In the field of artificial intelligence, problem solving can be simplified by an appropriate choice of *knowledge representation*. Representing knowledge in some ways makes certain problems easier to solve. For example, it is easier to divide numbers represented in Hindu-Arabic numerals than numbers represented as Roman numerals.

Characteristics

A good knowledge representation covers six basic characteristics:

- Coverage, which means the KR covers a breath and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
- Understandable by humans. KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (Polar bears are bears, which are animals). It should also have simple primitives that combine in complex forms.
- Consistency. If John closed the door, it can also be interpreted as the door was closed by John. By being consistent, the KR can eliminate redundant or conflicting knowledge.
- Efficient
- Easy to modify and update.
- Supports the intelligent activity which uses the knowledge base

To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopedia is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

History of knowledge representation and reasoning

In computer science, particularly artificial intelligence, a number of representations have been devised to structure information.

KR is most commonly used to refer to representations intended for processing by modern computers, and in particular, for representations consisting of explicit objects (the class of all elephants, or Clyde a certain individual), and of assertions or claims about them ('Clyde is an elephant', or 'all elephants are grey'). Representing knowledge in such explicit form enables computers to draw conclusions from knowledge already stored ('Clyde is grey').

Many KR methods were tried in the 1970s and early 1980s, such as heuristic question-answering, neural networks, theorem proving, and expert systems, with varying success. Medical diagnosis (e.g., Mycin) was a major application area, as were games such as chess.

In the 1980s formal computer knowledge representation languages and systems arose. Major projects attempted to encode wide bodies of general knowledge; for example the

"Cyc" project (still ongoing) went through a large encyclopedia, encoding not the information itself, but the information a reader would need in order to understand the encyclopedia: naive physics; notions of time, causality, motivation; commonplace objects and classes of objects.

Through such work, the difficulty of KR came to be better appreciated. In computational linguistics, meanwhile, much larger databases of language information were being built, and these, along with great increases in computer speed and capacity, made deeper KR more feasible.

Several programming languages have been developed that are oriented to KR. Prolog developed in 1972, but popularized much later, represents propositions and basic logic, and can derive conclusions from known premises. KL-ONE (1980s) is more specifically aimed at knowledge representation itself. In 1995, the Dublin Core standard of metadata was conceived.

In the electronic document world, languages were being developed to represent the structure of documents, such as SGML (from which HTML descended) and later XML. These facilitated information retrieval and data mining efforts, which have in recent years begun to relate to knowledge representation.

Development of the Semantic Web, has included development of XML-based knowledge representation languages and standards, including RDF, RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

Topics in Knowledge representation and reasoning

Language and notation

Some think it is best to represent knowledge in the same way that it is represented in the human mind, or to represent knowledge in the form of human language.

Psycholinguistics investigates how the human mind stores and manipulates language. Other branches of cognitive science examine how human memory stores sounds, sights, smells, emotions, procedures, and abstract ideas. Science has not yet completely described the internal mechanisms of the brain to the point where they can simply be replicated by computer programmers.

Various artificial languages and notations have been proposed for representing knowledge. They are typically based on logic and mathematics, and have easily parsed grammars to ease machine processing. They usually fall into the broad domain of ontologies.

Ontology Engineering

After CycL, a number of ontology languages have been developed. Most are declarative languages, and are either frame languages, or are based on first-order logic. Most of these languages only define an upper ontology with generic concepts, whereas the domain concepts are not part of the language definition. These languages all use special-purpose knowledge engineering because as stated by Tom Gruber, "Every ontology is a treaty- a social agreement among people with common motive in sharing." There are always many competing and differing views that make any general purpose ontology impossible. A general purpose ontology would have to be applicable in any domain and different areas of knowledge need to be unified. Gellish English is an example of an ontological language that includes a full engineering English Dictionary.

There is a long history of work attempting to build good ontologies for a variety of task domains, including early work on an ontology for liquids , the lumped element model widely used in representing electronic circuits (e.g.,), as well as ontologies for time, belief, and even programming itself. Each of these offers a way to see some part of the world. The lumped element model, for instance, suggests that we think of circuits in terms of components with connections between them, with signals flowing instantaneously along the connections. This is a useful view, but not the only possible one. A different ontology arises if we need to attend to the electrodynamics in the device: Here signals propagate at finite speed and an object (like a resistor) that was previously viewed as a single component with an I/O behavior may now have to be thought of as an extended medium through which an electromagnetic wave flows.

Ontologies can of course be written down in a wide variety of languages and notations (e.g., logic, LISP, etc.); the essential information is not the form of that language but the content, i.e., the set of concepts offered as a way of thinking about the world. Simply put, the important part is notions like connections and components, not whether we choose to write them as predicates or LISP constructs.

The commitment we make by selecting one or another ontology can produce a sharply different view of the task at hand. Consider the difference that arises in selecting the lumped element view of a circuit rather than the electrodynamic view of the same device. As a second example, medical diagnosis viewed in terms of rules (e.g., MYCIN) looks substantially different from the same task viewed in terms of frames (e.g., INTERNIST). Where MYCIN sees the medical world as made up of empirical associations connecting symptom to disease, INTERNIST sees a set of prototypes, in particular prototypical diseases, to be matched against the case at hand.

Commitment begins with the earliest choices

The INTERNIST example also demonstrates that there is significant and unavoidable ontological commitment even at the level of the familiar representation technologies. Logic, rules, frames, etc., each embody a viewpoint on the kinds of things that are important in the world. Logic, for instance, involves a (fairly minimal) commitment to viewing the world in terms of individual entities and relations between them. Rule-based systems view the world in terms of attribute-object-value triples and the rules of plausible inference that connect them, while frames have us thinking in terms of prototypical objects. Each of these thus supplies its own view of what is important to attend to, and each suggests, conversely, that anything not easily seen in those terms may be ignored. This is of course not guaranteed to be correct, since anything ignored may later prove to be relevant. But the task is hopeless in principle--every representation ignores something about the world--hence the best we can do is start with a good guess. The existing representation technologies supply one set of guesses about what to attend to and what to ignore. Selecting any of them thus involves a degree of ontological commitment: the selection will have a significant impact on our perception of and approach to the task, and on our perception of the world being modeled.

The commitments accumulate in layers

The ontologic commitment of a representation thus begins at the level of the representation technologies and accumulates from there. Additional layers of commitment are made as we put the technology to work. The use of frame-like structures in INTERNIST offers an illustrative example. At the most fundamental level, the decision to view diagnosis in terms of frames suggests thinking in terms of prototypes, defaults, and a taxonomic hierarchy. But prototypes of what, and how shall the taxonomy be organized? An early description of the system shows how these questions were answered in the task at hand, supplying the second layer of commitment:

The knowledge base underlying the INTERNIST system is composed of two basic types of elements: disease entities and manifestations.... [It] also contains a...hierarchy of disease categories, organized primarily around the concept of organ systems, having at the top level such categories as "liver disease," "kidney disease," etc.

The prototypes are thus intended to capture prototypical diseases (e.g., a "classic case" of a disease), and they will be organized in a taxonomy indexed around organ systems. This is a sensible and intuitive set of choices but clearly not the only way to apply frames to the task; hence it is another layer of ontological commitment.
At the third (and in this case final) layer, this set of choices is instantiated: which diseases will be included and in which branches of the hierarchy will they appear? Ontologic questions that arise even at this level can be quite fundamental. Consider for example determining which of the following are to be considered diseases (i.e., abnormal states requiring cure): alcoholism, homosexuality, and chronic fatigue syndrome. The ontologic commitment here is sufficiently obvious and sufficiently important that it is often a subject of debate in the field itself, quite independent of building automated reasoners.
Similar sorts of decisions have to be made with all the representation technologies, because each of them supplies only a first order guess about how to see the world: they offer a way of seeing but don't indicate how to instantiate that view. As frames suggest prototypes and taxonomies but do not tell us which things to select as prototypes, rules suggest thinking in terms of plausible inferences, but don't tell us which plausible inferences to attend to. Similarly logic tells us to view the world in terms of individuals and relations, but does not specify which individuals and relations to use.
Commitment to a particular view of the world thus starts with the choice of a

representation technology, and accumulates as subsequent choices are made about how to see the world in those terms.

Reminder: A KR is not a data structure
Note that at each layer, even the first (e.g., selecting rules or frames), the choices being made are about representation, not data structures. Part of what makes a language representational is that it carries meaning , i.e., there is a correspondence between its constructs and things in the external world. That correspondence in turn carries with it constraint. A semantic net, for example, is a representation, while a graph is a data structure. They are different kinds of entities, even though one is invariably used to implement the other, precisely because the net has (should have) a semantics. That semantics will be manifest in part because it constrains the network topology: a network purporting to describe family memberships as we know them cannot have a cycle in its parent links, while graphs (i.e., data structures) are of course under no such constraint and may have arbitrary cycles.
While every representation must be implemented in the machine by some data structure, the representational property is in the correspondence to something in the world and in the constraint that

Links and structures

While hyperlinks have come into widespread use, the closely related semantic link is not yet widely used. The mathematical table has been used since Babylonian times. More recently, these tables have been used to represent the outcomes of logic operations, such as truth tables, which were used to study and model Boolean logic, for example. Spreadsheets are yet another tabular representation of knowledge. Other knowledge representations are trees, graphs and hypergraphs, by means of which the connections among fundamental concepts and derivative concepts can be shown.

Visual representations are relatively new in the field of knowledge management but give the user a way to visualise how one thought or idea is connected to other ideas enabling the possibility of moving from one thought to another in order to locate required information.

Notation

The recent fashion in knowledge representation languages is to use XML as the low-level syntax. This tends to make the output of these KR languages easy for machines to parse, at the expense of human readability and often space-efficiency.

First-order predicate calculus is commonly used as a mathematical basis for these systems, to avoid excessive complexity. However, even simple systems based on this simple logic can be used to represent data that is well beyond the processing capability of current computer systems.

Examples of notations:

- DATR is an example for representing lexical knowledge
- RDF is a simple notation for representing relationships between and among objects

Storage and manipulation

One problem in knowledge representation is how to store and manipulate knowledge in an information system in a formal way so that it may be used by mechanisms to accomplish a given task. Examples of applications are expert systems, machine translation systems, computer-aided maintenance systems and information retrieval systems (including database front-ends).

Semantic networks may be used to represent knowledge. Each node represents a concept and arcs are used to define relations between the concepts.The Conceptual graph model is probably the oldest model still alive. One of the most expressive and comprehensively described knowledge representation paradigms along the lines of semantic networks is MultiNet (an acronym for Multilayered Extended Semantic Networks).

From the 1960s, the knowledge frame or just *frame* has been used. Each frame has its own name and a set of **attributes**, or **slots** which contain values; for instance, the frame for *house* might contain a *color* slot, *number of floors* slot, etc.

Using frames for expert systems is an application of object-oriented programming, with inheritance of features described by the "is-a" link. However, there has been no small amount of inconsistency in the usage of the "is-a" link: Ronald J. Brachman wrote a paper titled "What IS-A is and isn't", wherein 29 different semantics were found in projects whose knowledge representation schemes involved an "is-a" link. Other links include the "part-of" link.

Frame structures are well-suited for the representation of schematic knowledge and stereotypical cognitive patterns. The elements of such schematic patterns are weighted unequally, attributing higher weights to the more typical elements of a schema. A pattern is activated by certain expectations: If a person sees a big bird, he or she will classify it rather as a sea eagle than a golden eagle, assuming that his or her "sea-scheme" is currently activated and his "land-scheme" is not.

Frame representations are object-centered in the same sense as semantic networks are: All the facts and properties connected with a concept are located in one place - there is no need for costly search processes in the database.

A behavioral script is a type of frame that describes what happens temporally; the usual example given is that of describing going to a restaurant. The steps include waiting to be seated, receiving a menu, ordering, etc. The different solutions can be arranged in a so-called semantic spectrum with respect to their semantic expressivity.

Chapter-8

Universal Decimal Classification

The **Universal Decimal Classification** is a system of library classification developed by the Belgian bibliographers Paul Otlet and Henri La Fontaine at the end of the 19th century. It is based on the Dewey Decimal Classification, but uses auxiliary signs to indicate various special aspects of a subject and relationships between subjects. It thus contains a significant faceted or analytico-synthetic element, and is used especially in specialist libraries. UDC has been modified and extended through the years to cope with the increasing output in all disciplines of human knowledge, and is still under continuous review to take account of new developments.

The documents classified by UDC may be in any form. They will often be literature, i.e. written documents, but may also be in other media such as films, video and sound recordings, illustrations, maps, and realia such as museum pieces.

UDC classifications use Arabic numerals and are based on the decimal system. Every number is thought of as a decimal fraction with the initial decimal point omitted, which determines filing order. For ease of reading, a UDC identifier is usually punctuated after every third digit. Thus, after 61 "Medical sciences" come the subdivisions 611 to 619; under 611 "Anatomy" come its subdivisions 611.1 to 611.9; under 611.1 come all of its subdivisions before 611.2 occurs, and so on; after 619 comes 620. An advantage of this system is that it is infinitely extensible, and when new subdivisions are introduced, they need not disturb the existing allocation of numbers.

The main categories

- 0 generalities
- 1 philosophy, psychology
- 2 religion, theology
- 3 social sciences
- 4
- 5 natural sciences

- 6 technology
- 7 the arts
- 8 language, linguistics, literature
- 9 geography, biography, history

A document may be classified under a combination of different categories through the use of additional symbols. For example:

Symbol	Symbol name	Meaning	Example
+	plus	addition	e.g. 59+636 zoology and animal breeding
/	stroke	extension	e.g. 592/599 Systematic zoology (everything from 592 to 599 inclusive)
:	colon	relation	e.g. 17:7 Relation of ethics to art
[]	square brackets	algebraic subgrouping	e.g. 311:[622+669](485) statistics of mining and metallurgy in Sweden (the auxiliary qualifies 622+669 considered as a unit)
=	equals	language	e.g. =111 in English; 59=111 Zoology, in English

The design of UDC lends itself to machine readability, and the system has been used both with early automatic mechanical sorting devices, and modern library OPACs. A core version of UDC, with 65,000 subdivisions, is now available in database format, and is called the Master Reference File (MRF). The current full version of the UDC has 220,000 subdivisions.

UDC Classes Table

Main Table

0 Generalities

```
 000 Computer science, knowledge & systems
   001          Science and knowledge in general. Organization of
intellectual work
   002  Documentation. Books. Writings. Authorship
   003  Writing systems and scripts. Including: signs and symbols
   004  Computer science and technology. Computing
     004.2      Computer architecture
     004.3      Computer hardware
     004.4      Software
     004.5      Human-computer interaction
     004.6      Data
     004.7      Computer communication
     004.8      Artificial intelligence
     004.9   Application-oriented computer-based techniques
   005  Management (Revision from 2001)
     005.1      Management Theory
     005.2      Management agents. Mechanisms. Measures
```

005.3 Management activities
005.32 Organizational behaviour. Management psychology
005.5 Management operations. Direction
005.6 Quality management. Total quality management (TQM)
005.7 Organizational management (OM)
005.9 Fields of management
005.92 Records management
005.93 Plant management. Physical resources management
005.94 Knowledge management
005.95/.96 Personnel management. Human Resources management
006 Standardization of products, operations, weights, measures and time
007 Activity and organizing. Information. Communication and control theory generally (cybernetics)
008 Civilization. Culture. Progress
009 Humanities. Arts subjects in general
010 Bibliographies
020 Library and information sciences
030 Encyclopedias & books of facts
040 [Unassigned]
050 Magazines, journals, periodicals & serials
060 Associations and organizations & museums
070 News media, journalism, Mass media & publishing
080 Quotations
090 Manuscripts & rare books

1 Philosophy. Psychology

100 Philosophy
110 Metaphysics
120 Epistemology
130 Parapsychology & ism
140 Philosophical schools of thought
159.9 Psychology
159.91 Psychophysiology (physiological psychology). Mental physiology
159.92 Mental development and capacity. Comparative psychology
159.93 Sensation. Sensory perception
159.94 Executive functions
159.95 Higher mental processes
159.96 Special mental states and processes
159.97 Abnormal psychology. Insanity. Mental deficiency
159.98 Applied psychology
160 Logic
170 Ethics
180 Ancient, medieval & eastern philosophy
190 Modern western philosophy

2 Religion. Theology

200 Religion
21 Prehistoric and primitive religions
22 Religions of the Far East
221 Religions of China
221.3 Taoism
223 Religions of Korea

```
      225        Religions of Japan
    23   Religions of the Indian subcontinent
      233        Hinduism narrowly
      234        Jainism
      235        Sikhism
    24   Buddhism
      241        Hinayana (Theravada) Buddhism
      242        Mahayana Buddhism
      243        Lamaism
      244        Japanese Buddhism
    25  Religions of antiquity. Minor cults and religions
      252        Religions of Mesopotamia
      254        Religions of Iran
      257        Religions of Europe
    26   Judaism
      262        Ashkenazi Judaism
      264        Sephardi Judaism
      265        Orthodox Judaism
      266        Progressive Judaism
      267        Modern movements arising from Judaism
    27   Christianity
      271        Eastern church
      272/279 Western church
        272      Roman Catholic church
        273      Non-Roman Catholic episcopal churches
        274      Protestantism generally. Protestants. Dissenters.
Puritans
        275      Reformed churches
        276      Anabaptists
        277      Free churches. Non-conformists
        278      Other protestant churches
        279      Other Christian movements and churches
    28   Islam
      282        Sunni. Sunnite Islam
      284        Shi'a. Shi'ite Islam
      285        Babi-Baha'i
      286        Baha'i
    29   Modern spiritual movements
 210 Philosophy & theory of religion
 220 Religions of the Far East
 230 Religions of the Indian subcontinent
 240 Buddhism
 250 Religions of antiquity. Minor cults and religions
 260 Judaism
 270 Christianity
 280 Islam
 290 Modern spiritual movements
```

3 Social Sciences

```
 300 Social sciences, sociology & anthropology
 310 Statistics
 320 Political science
 330 Economics
 340 Law
 350 Public administration & military science
 360 Social problems & social services
 370 Education
 380 Commerce, communications & transportation
 390 Customs, etiquette & folklore
```

5 Mathematics and natural sciences

```
 500 Science
 510 Mathematics
   510 Fundamental and general consideration of mathematics
   511 Number theory
   512 Algebra
   514 Geometry
   515.1 Topology
   517 Analysis
   519.1 Combinatorial analysis. Graph theory
 520 Astronomy
 530 Physics
   531 General mechanics. Mechanics of solid and rigid bodies
   532 Fluid mechanics in general. Mechanics of liquids
(hydromechanics)
   533 Mechanics of gases. Aeromechanics. Plasma physics
   534 Vibrations. Acoustics
   535 Optics
   536 Heat. Thermodynamics
   537 Electricity. Magnetism. Electromagnetism
   539 Physical nature of matter
 540 Chemistry
   542 Practical laboratory chemistry
   543 Analytical chemistry
   544 Physical chemistry
   546 Inorganic chemistry
   547 Organic chemistry
   548 Crystallography
 550 Earth sciences & geology
 560 Fossils & prehistoric life
 570 Life sciences; biology
 580 Plants (Botany)
 590 Animals (Zoology)
```

6 Applied sciences. Medicine. Technology

```
 600 Technology
 610 Medicine & health
   611 Medicine
   612 Pharmacy
   613 Biomedical Sciences
   614 Public Health
   615 Kinesitherapy - Physical Training
 620 Engineering. Technology in general
   620 Materials testing. Commercial materials. Power stations.
Economics of energy
   621 Mechanical engineering in general. Nuclear technology.
Electrical engineering. Machinery
   621.3 Electrical engineering
   622 Mining
   623 Military engineering
   624 Civil and structural engineering in general
```

625 Civil engineering and land transport. Railway engineering. Highway engineering
626 Hydraulic engineering in general
627 Natural waterway, port, harbour and shore engineering. Navigational, dredging, salvage and rescue facilities. Dams and hydraulic power plant
628 Public health engineering. Water. Sanitation. Illuminating engineering
629 Transport vehicle engineering
630 Agriculture
640 Home & family management
650 Management & public relations
660 Chemical engineering
670 Manufacturing
680 Manufacture for specific uses
690 Building & construction

7 The arts. Recreation. Entertainment. Sport

700 Arts
710 Landscaping & area planning
720 Architecture
730 Sculpture, ceramics & metalwork
740 Drawing & decorative arts
750 Painting
760 Graphic arts
770 Photography & computer art
780 Music
790 Sports, games & entertainment

8 Language. Linguistics. Literature

800 General questions. Including: Philology. Rhetoric
810 Linguistics and languages
820 Literature

9 Geography. Biography. History

900 History
910 Geography & travel
920 Biography & genealogy
930 History of ancient world (to ca. 499)
940 History of Europe
950 History of Asia
960 History of Africa
970 History of North America
980 History of South America
990 History of other areas

Table 2 (Geographic Areas, Historical Periods, Persons)

-1 Place in general
-2 Physiographic designation
-3 The ancient world
-4 Europe
-41 British Isles (geographical whole)
-410 United Kingdom of Gt Britain and N Ireland
-410.1 England

-410.3 Wales
-410.5 Scotland
-410.7 Northern Ireland
-415 Ireland (geographical whole)
-417 Republic of Ireland
-430 Germany
-436 Austria
-437.1 Czech Republic
-437.6 Slovak Republic
-438 Poland
-439 Hungary
-44 France
-450 Italy
-4549 San Marino
-45634 Vatican City
-4585 Malta
-460 Spain
-469 Portugal
-47 Former European USSR
-470 Russia
-48 Scandinavia
-480 Finland
-481 Norway
-485- Sweden
-489 Denmark
-492 Netherlands
-493 Belgium
-494 Switzerland
-495 Greece
-497 Balkan States
-5 Asia
-61 Tunisia, Libya
-7 North and Central America
-71 Canada
-72 Mexico
-728 Central America
-729 West Indies
-73 USA
-74 N E States
-75 S E States
-76 S Central States
-77 N Central States
-78 W States
-79 Pacific States
-8 South America
-9 South Pacific and Australia. Arctic. Antarctic

Table 5 (Ethnic and National Groups)

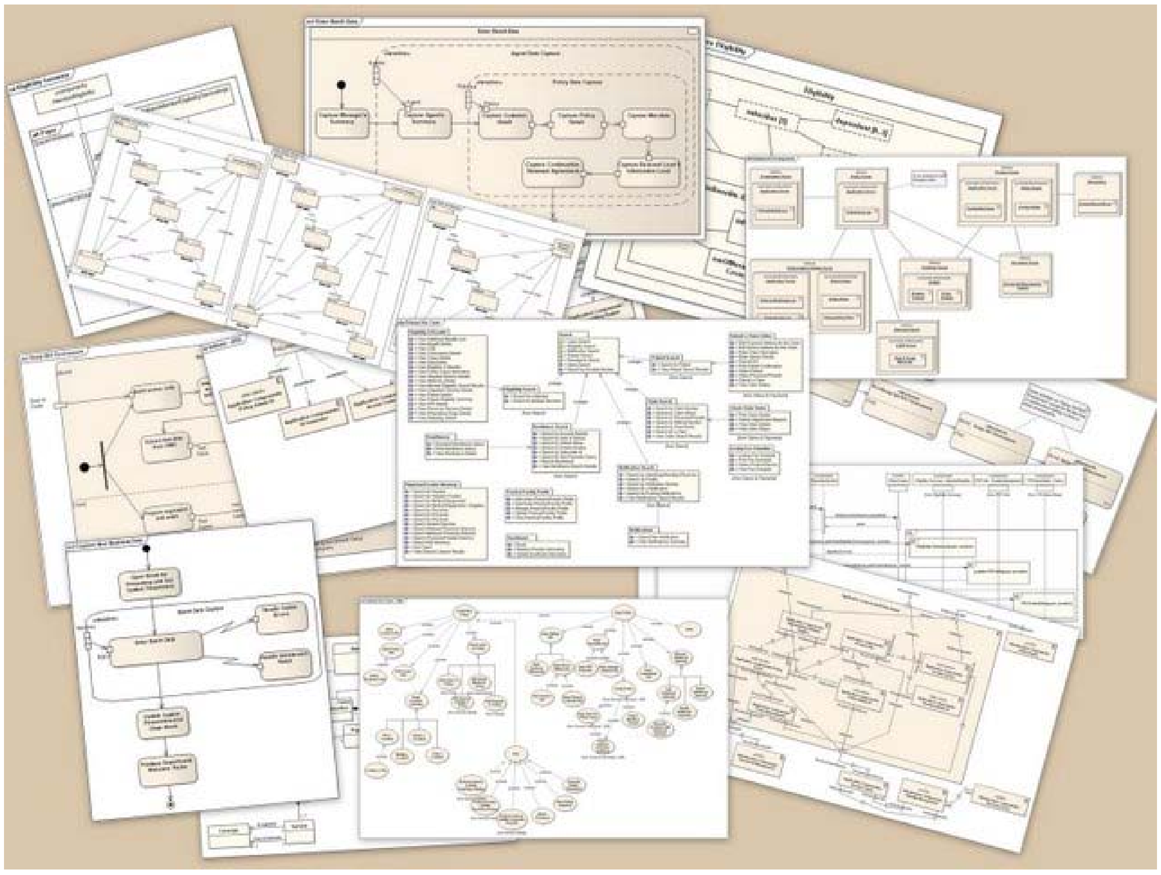
-5 Italians, Romanians, related groups
-591 Romanians
-5994 Ladins
-5998 Sardinians and Corsicans
-59982 Sardinians
-59984 Corsicans

Table 6 (Languages)

-21	English language
-59	Romanian, Rhaetian, Sardinian, Corsican
-5992	Friulian language
-5994	Ladin language
-5996	Romansch language
-59982	Sardinian
-59984	Corsican
-67	Judeo-Spanish (Ladino)
-9455	Sami

Chapter-9

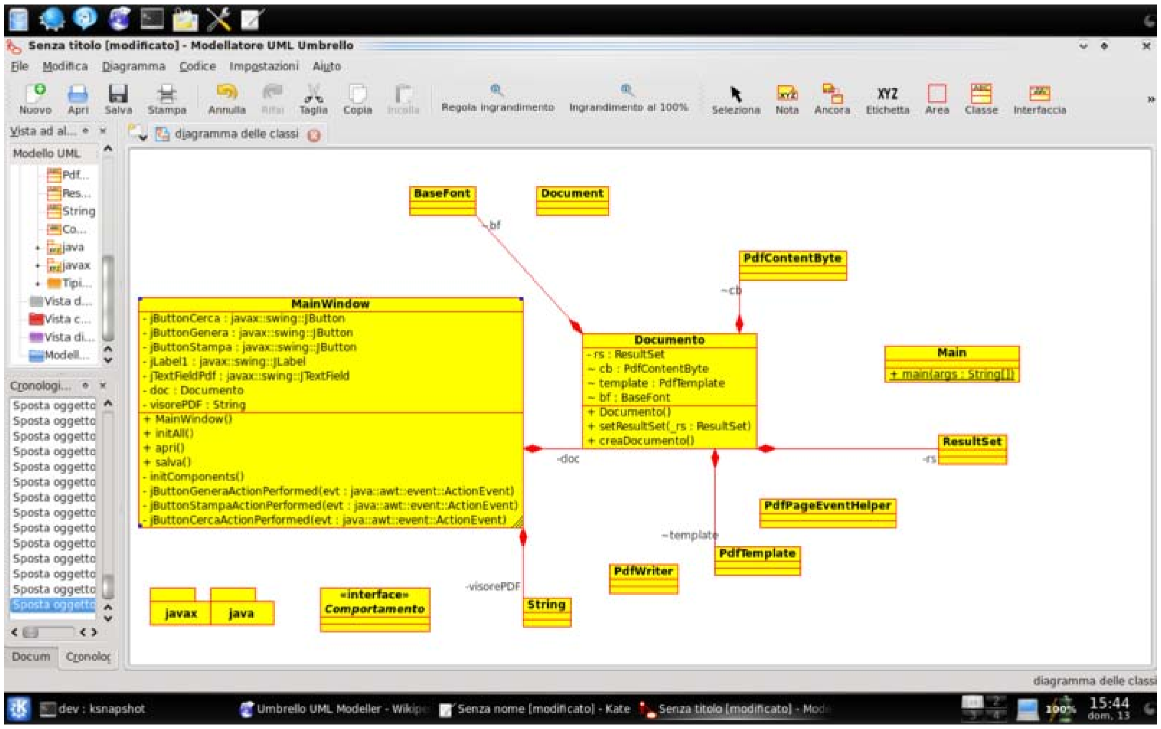
Unified Modeling Language



A collage of UML diagrams.

Unified Modeling Language (**UML**) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.


Senza titolo [modificato] - Modellatore UML Umbrello
File Modifica Diagramma Codice Impostazioni Aiuto
Nuovo Apri Salva Stampa Annulla Rifai Taglia Copia Incolla Regola ingrandimento Ingrandimento al 100% Seleziona Nota Ancora Etichetta Area Classe Interfaccia
diagramma delle classi
BaseFont
Document
PdfContentByte
MainWindow
- jButtonCerca : javax::swing::JButton
- jButtonGenera : javax::swing::JButton
- jButtonStampa : javax::swing::JButton
- jLabel1 : javax::swing::JLabel
- jTextFieldPdf : javax::swing::JTextField
- doc : Documento
- visorePDF : String
+ MainWindow()
+ initAll()
+ apri()
+ salva()
- initComponents()
- jButtonGeneraActionPerformed(evt : java::awt::event::ActionEvent)
- jButtonStampaActionPerformed(evt : java::awt::event::ActionEvent)
- jButtonCercaActionPerformed(evt : java::awt::event::ActionEvent)
Documento
- rs : ResultSet
~ cb : PdfContentByte
~ template : PdfTemplate
~ bf : BaseFont
+ Documento()
+ setResultSet(_rs : ResultSet)
+ creaDocumento()
Main
+ main(args : String[])
ResultSet
PdfPageEventHelper
PdfTemplate
PdfWriter
String
«interface» Comportamento
javax
java
~bf
~cb
-doc
-rs
~template
-visorePDF

Screenshot of Umbrello UML Modeller.

Overview

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

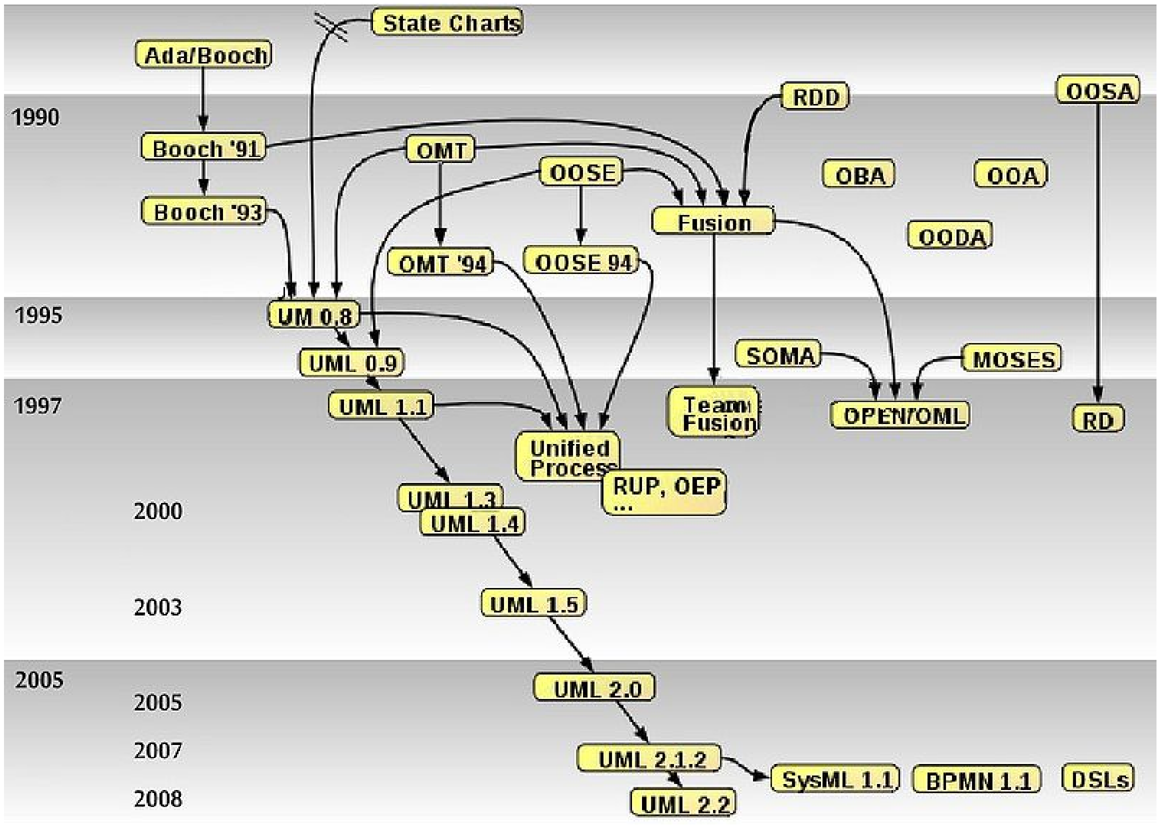
- activities
- actors
- business processes
- database schemas
- (logical) components
- programming language statements
- reusable software components.

UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

UML is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG).

UML models may be automatically transformed to other representations (e.g. Java) by means of QVT-like transformation languages. UML is extensible, with two mechanisms for customization: profiles and stereotypes.

History



History of object-oriented methods and notation.

Before UML 1.x

After Rational Software Corporation hired James Rumbaugh from General Electric in 1994, the company became the source for the two most popular object-oriented modeling approaches of the day: Rumbaugh's Object-modeling technique (OMT), which was better for object-oriented analysis (OOA), and Grady Booch's Booch method, which was better for object-oriented design (OOD). They were soon assisted in their efforts by Ivar Jacobson, the creator of the object-oriented software engineering (OOSE) method. Jacobson joined Rational in 1995, after his company, Objectory AB, was acquired by Rational. The three methodologists were collectively referred to as the *Three Amigos*.

In 1996 Rational concluded that the abundance of modeling languages was slowing the adoption of object technology, so repositioning the work on an unified method, they tasked the Three Amigos with the development of a non-proprietary Unified Modeling Language. Representatives of competing object technology companies were consulted during OOPSLA '96; they chose *boxes* for representing classes rather than the *cloud* symbols that were used in Booch's notation.

Under the technical leadership of the Three Amigos, an international consortium called the UML Partners was organized in 1996 to complete the *Unified Modeling Language (UML)* specification, and propose it as a response to the OMG RFP. The UML Partners' UML 1.0 specification draft was proposed to the OMG in January 1997. During the same month the UML Partners formed a Semantics Task Force, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the semantics of the specification and integrate it with other standardization efforts. The result of this work, UML 1.1, was submitted to the OMG in August 1997 and adopted by the OMG in November 1997.

UML 1.x

As a modeling notation, the influence of the OMT notation dominates (e. g., using rectangles for classes and objects). Though the Booch "cloud" notation was dropped, the Booch capability to specify lower-level design detail was embraced. The use case notation from Objectory and the component notation from Booch were integrated with the rest of the notation, but the semantic integration was relatively weak in UML 1.1, and was not really fixed until the UML 2.0 major revision.

Concepts from many other OO methods were also loosely integrated with UML with the intent that UML would support all OO methods. Many others also contributed, with their approaches flavouring the many models of the day, including: Tony Wasserman and Peter Pircher with the "Object-Oriented Structured Design (OOSD)" notation (not a method), Ray Buhr's "Systems Design with Ada", Archie Bowen's use case and timing analysis, Paul Ward's data analysis and David Harel's "Statecharts"; as the group tried to ensure broad coverage in the real-time systems domain. As a result, UML is useful in a variety of engineering problems, from single process, single user applications to concurrent, distributed systems, making UML rich but also large.

The Unified Modeling Language is an international standard:

> ISO/IEC 19501:2005 Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2

UML 2.x

UML has matured significantly since UML 1.1. Several minor revisions (UML 1.3, 1.4, and 1.5) fixed shortcomings and bugs with the first version of UML, followed by the UML 2.0 major revision that was adopted by the OMG in 2005.

Although UML 2.1 was never released as a formal specification, versions 2.1.1 and 2.1.2 appeared in 2007, followed by UML 2.2 in February 2009. UML 2.3 was formally released in May 2010.

There are four parts to the UML 2.x specification:

1. The Superstructure that defines the notation and semantics for diagrams and their model elements
2. The Infrastructure that defines the core metamodel on which the Superstructure is based
3. The Object Constraint Language (OCL) for defining rules for model elements
4. The UML Diagram Interchange that defines how UML 2 diagram layouts are exchanged

The current versions of these standards follow: UML Superstructure version 2.3, UML Infrastructure version 2.3, OCL version 2.2, and UML Diagram Interchange version 1.0.

Although many UML tools support some of the new features of UML 2.x, the OMG provides no test suite to objectively test compliance with its specifications.

Topics

Software development methods

UML is not a development method by itself; however, it was designed to be compatible with the leading object-oriented software development methods of its time (for example OMT, Booch method, Objectory). Since UML has evolved, some of these methods have been recast to take advantage of the new notations (for example OMT), and new methods have been created based on UML, such as IBM Rational Unified Process (RUP). Others include Abstraction Method and Dynamic Systems Development Method.

Modeling

It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drive the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

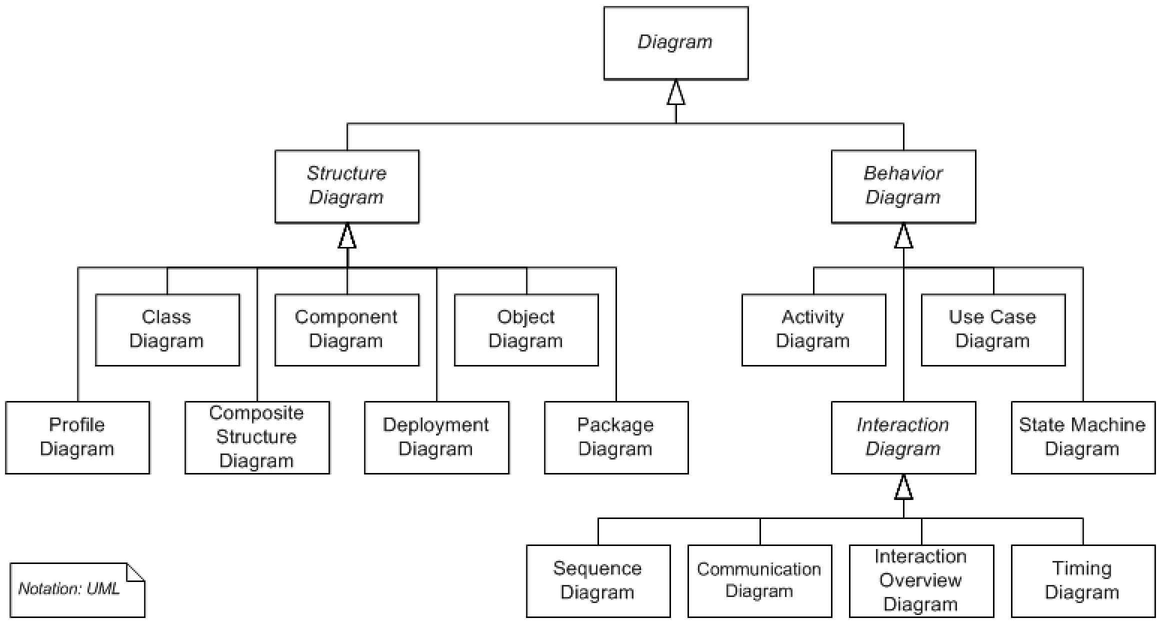
- Static (or *structural*) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
- Dynamic (or *behavioral*) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of

objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

UML models can be exchanged among UML tools by using the XMI interchange format.

Diagrams overview

UML 2.2 has 14 types of diagrams divided into two categories. Seven diagram types represent *structural* information, and the other seven represent general types of *behavior*, including four that represent different aspects of *interactions*. These diagrams can be categorized hierarchically as shown in the following class diagram:



UML does not restrict UML element types to a certain diagram type. In general, every UML element may appear on almost all types of diagrams; this flexibility has been partially restricted in UML 2.0. UML profiles may define additional diagram types or extend existing diagrams with additional notations.

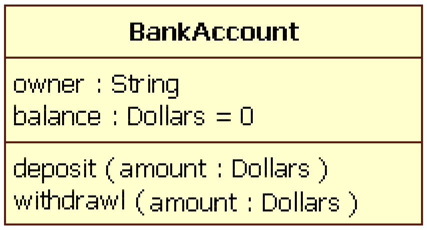
In keeping with the tradition of engineering drawings, a comment or note explaining usage, constraint, or intent is allowed in a UML diagram.

Structure diagrams

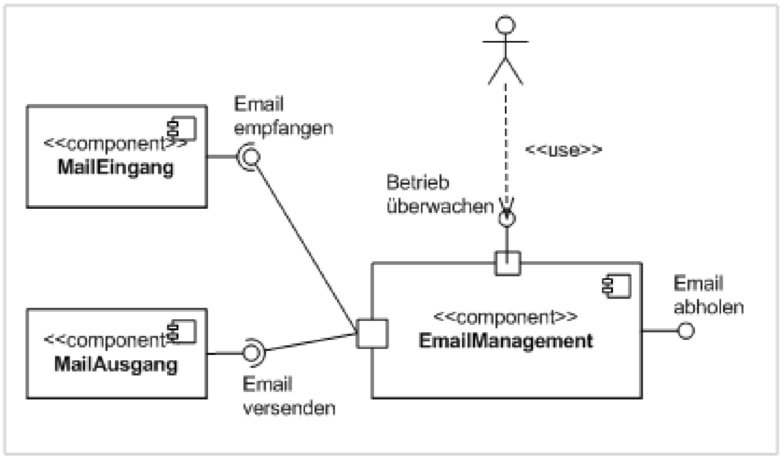
Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

- Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.

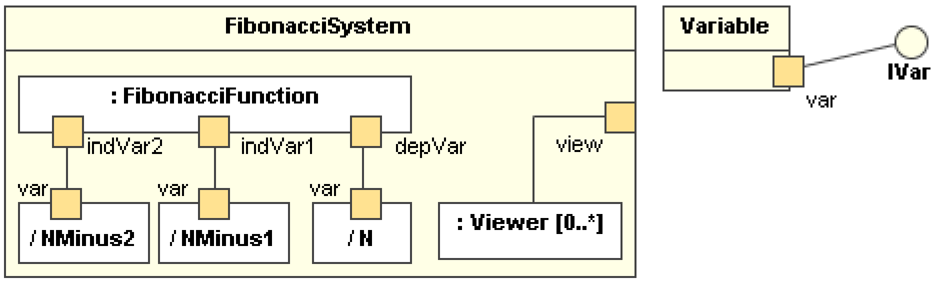
- Component diagram: describes how a software system is split up into components and shows the dependencies among these components.
- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of a modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.



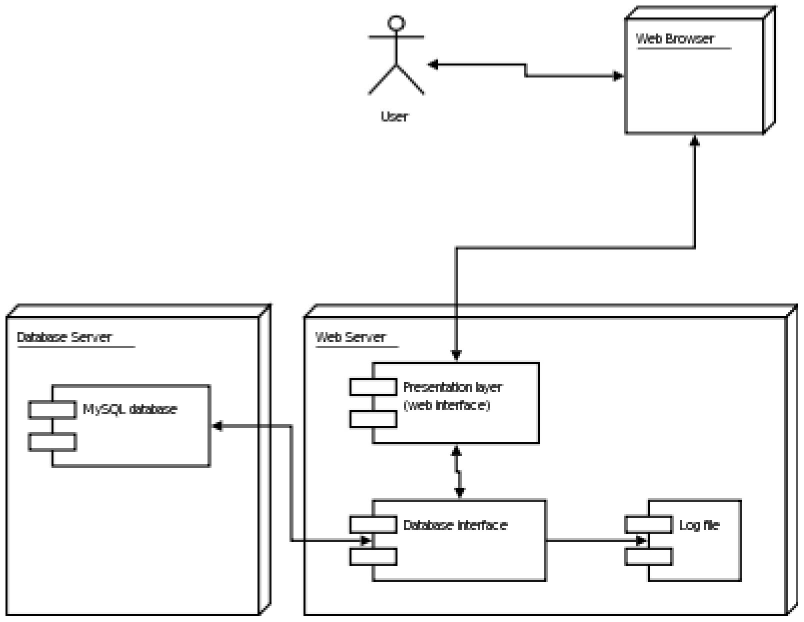
Class diagram



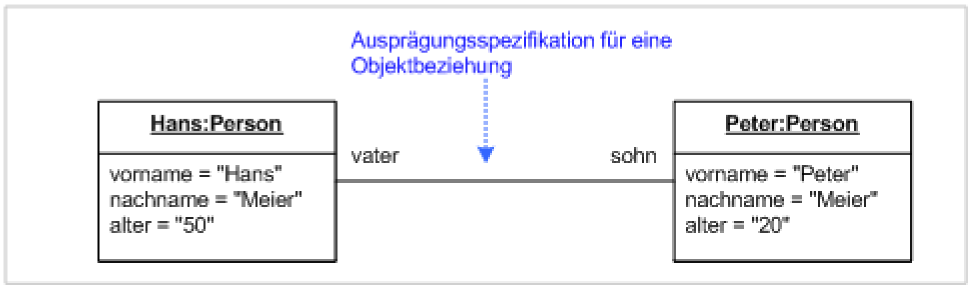
Component diagram



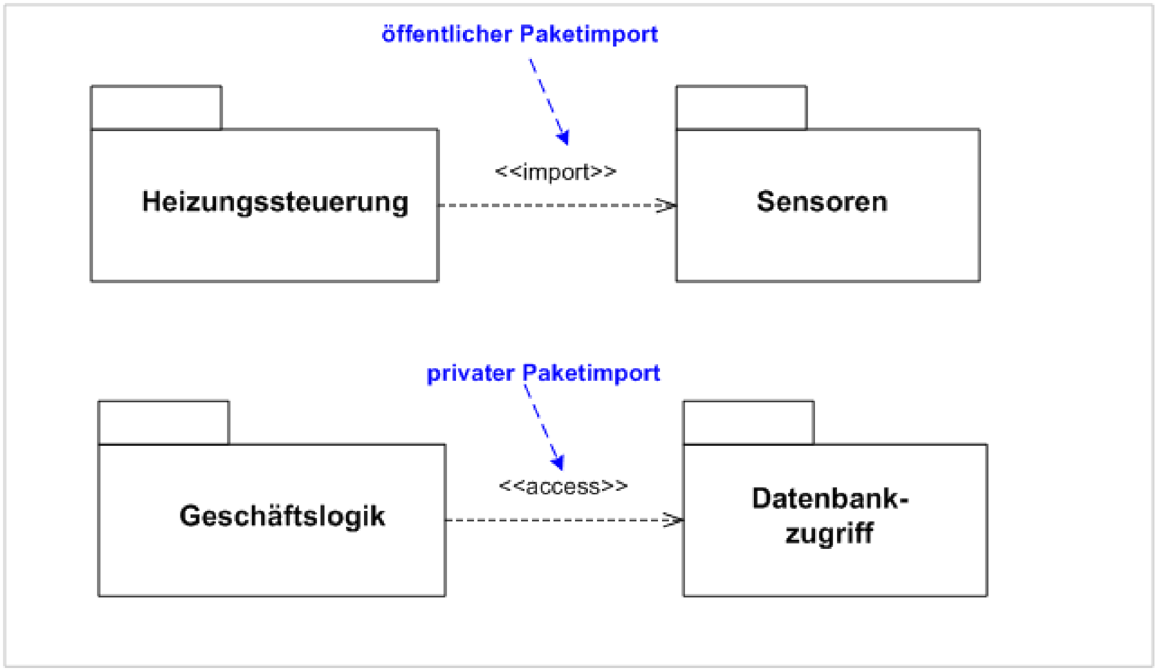
Composite structure diagrams



Deployment diagram



Object diagram

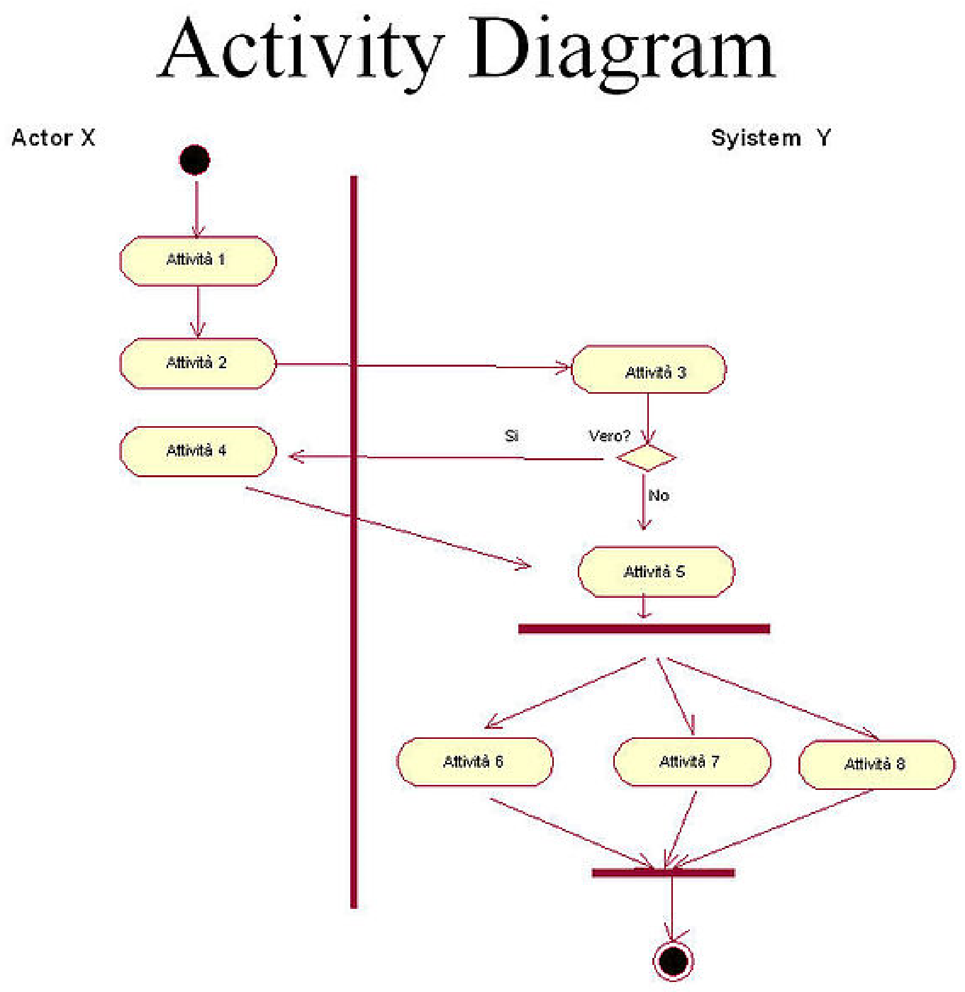


Package diagram

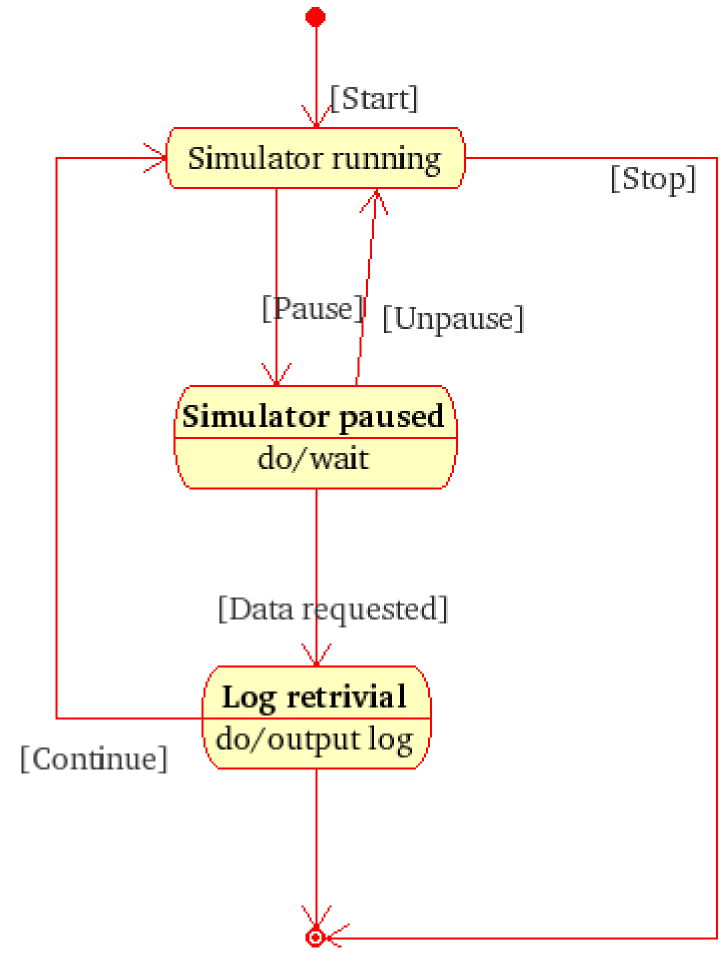
Behaviour diagrams

Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

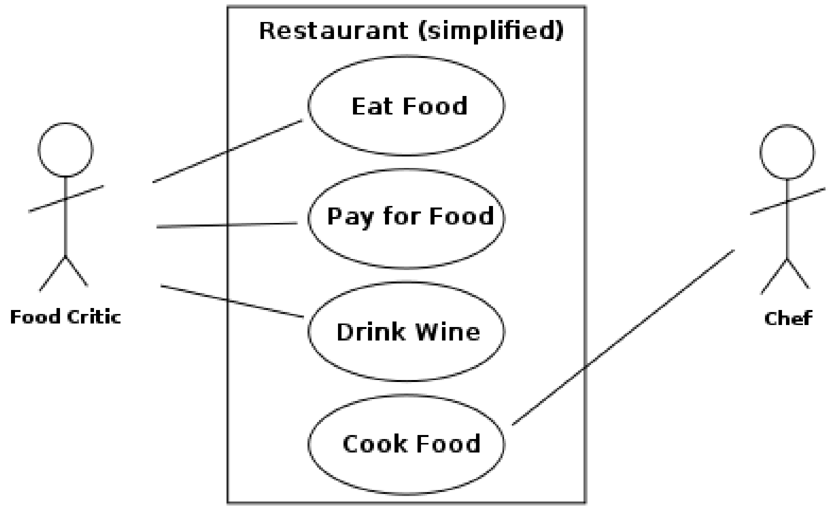
- Activity diagram: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- UML state machine diagram: describes the states and state transitions of the system.
- Use case diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.



UML Activity Diagram



State Machine diagram

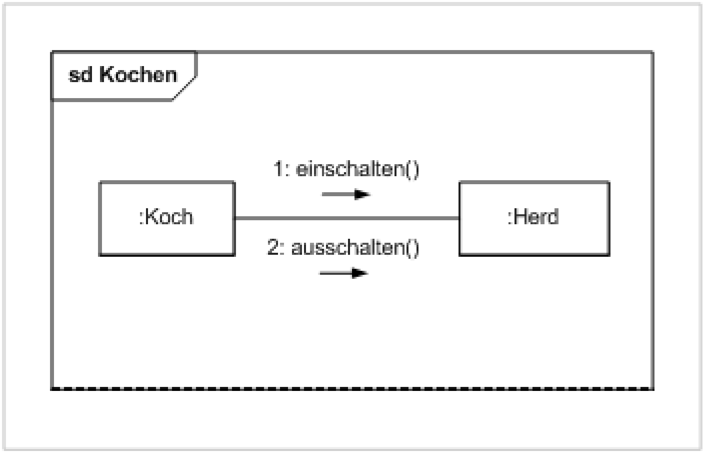


Use case diagram

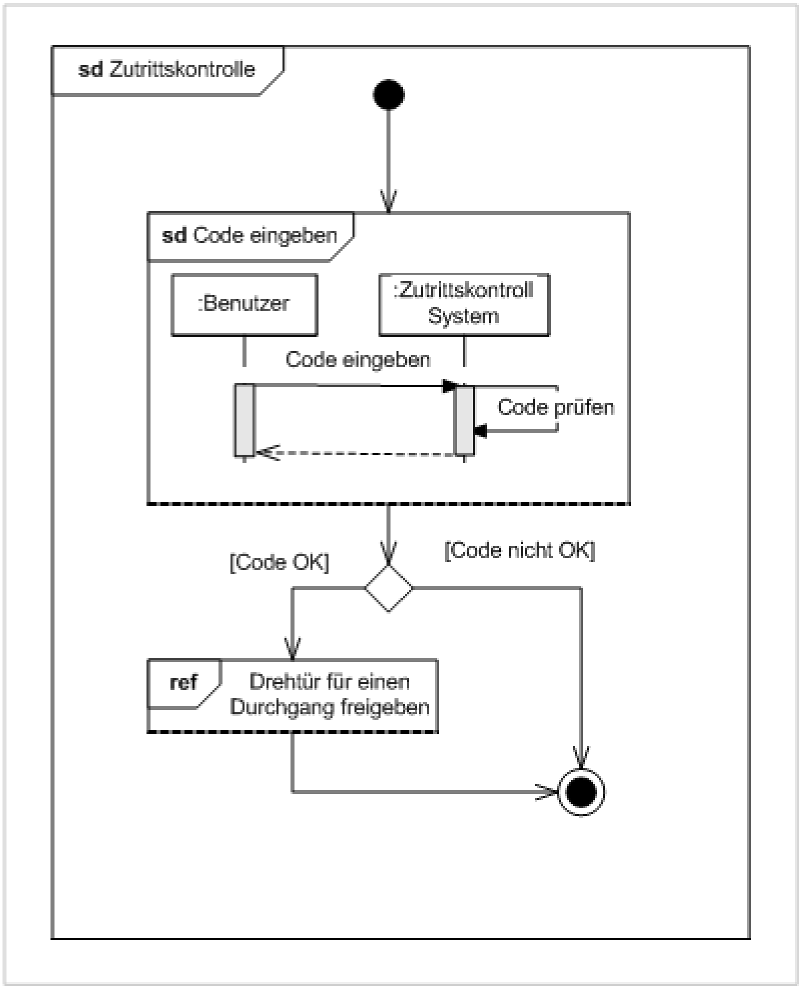
Interaction diagrams

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modeled:

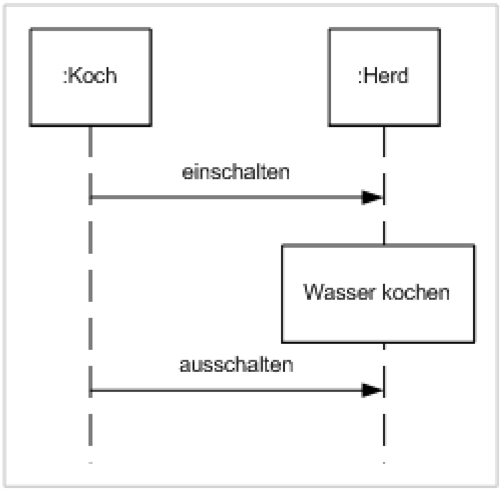
- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.
- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.



Communication diagram



Interaction overview diagram



Sequence diagram

The Protocol State Machine is a sub-variant of the State Machine. It may be used to model network communication protocols.

Meta modeling

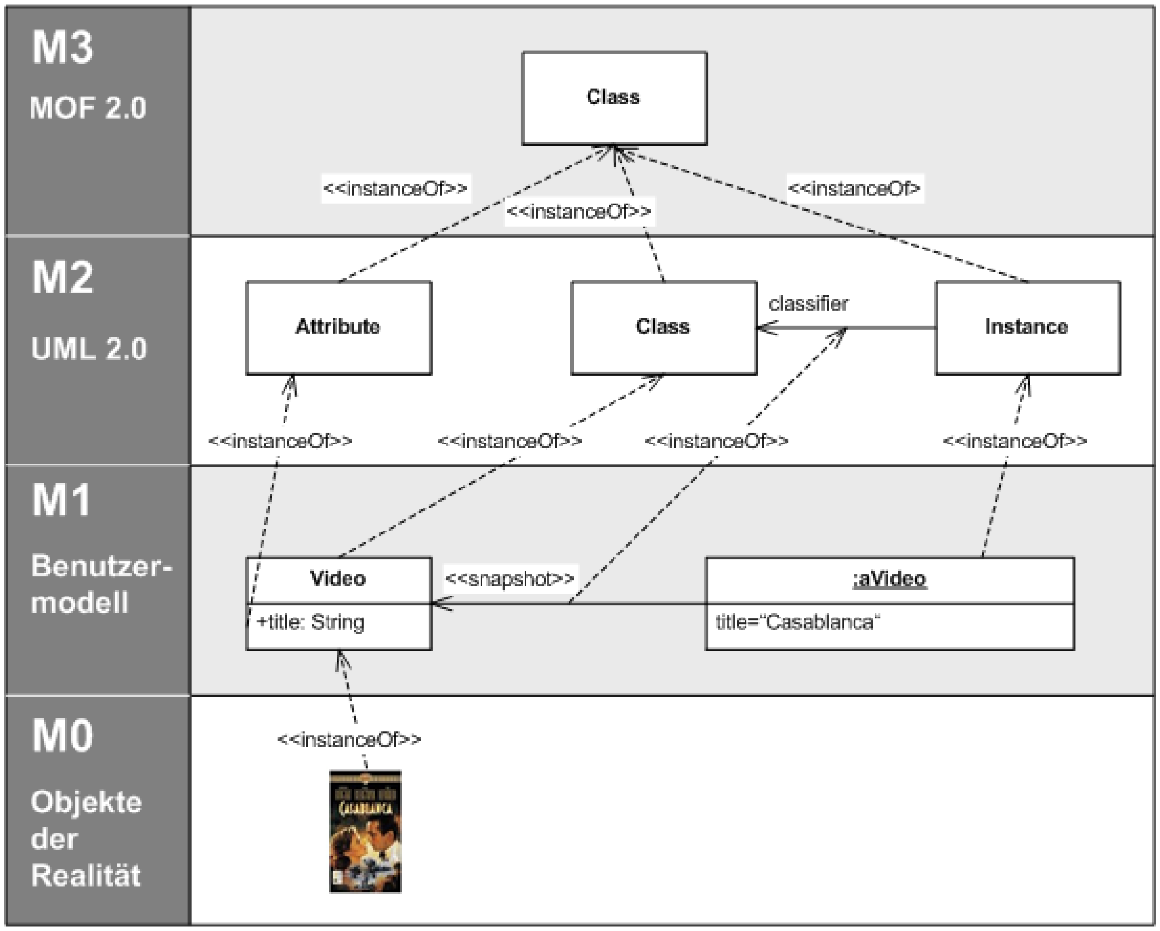


Illustration of the Meta-Object Facility.

The Object Management Group (OMG) has developed a metamodeling architecture to define the Unified Modeling Language (UML), called the Meta-Object Facility (MOF). The Meta-Object Facility is a standard for model-driven engineering, designed as a four-layered architecture, as shown in the image at right. It provides a meta-meta model at the top layer, called the M0 layer. This M0-model is the language used by Meta-Object Facility to build metamodels, called M1-models. The most prominent example of a Layer 1 Meta-Object Facility model is the UML metamodel, the model that describes the UML itself. These M1-models describe elements of the M2-layer, and thus M2-models. These would be, for example, models written in UML. The last layer is the M3-layer or data layer. It is used to describe runtime instance of the system.

Beyond the M0-model, the Meta-Object Facility describes the means to create and manipulate models and metamodels by defining CORBA interfaces that describe those operations. Because of the similarities between the Meta-Object Facility M0-model and

UML structure models, Meta-Object Facility metamodels are usually modeled as UML class diagrams. A supporting standard of the Meta-Object Facility is XMI, which defines an XML-based exchange format for models on the M0-, M1-, or M2-Layer.

Criticisms

Although UML is a widely recognized and used modeling standard, it is frequently criticized for the following:

Standards bloat

Bertrand Meyer, in a satirical essay framed as a student's request for a grade change, apparently criticized UML as of 1997 for being unrelated to object-oriented software development; a disclaimer was added later pointing out that his company nevertheless supports UML. Ivar Jacobson, a co-architect of UML, said that objections to UML 2.0's size were valid enough to consider the application of intelligent agents to the problem. It contains many diagrams and constructs that are redundant or infrequently used.

Problems in learning and adopting

The problems cited here make learning and adopting UML problematic, especially when required of engineers lacking the prerequisite skills. In practice, people often draw diagrams with the symbols provided by their CASE tool, but without the meanings those symbols are intended to provide.

Linguistic incoherence

The extremely poor writing of the UML standards themselves—assumed to be the consequence of having been written by a non-native English speaker—seriously reduces their normative value. In this respect the standards have been widely cited, and indeed pilloried, as prime examples of unintelligible geekspeak.

Capabilities of UML and implementation language mismatch

As with any notational system, UML is able to represent some systems more concisely or efficiently than others. Thus a developer gravitates toward solutions that reside at the intersection of the capabilities of UML and the implementation language. This problem is particularly pronounced if the implementation language does not adhere to orthodox object-oriented doctrine, as the intersection set between UML and implementation language may be that much smaller.

Dysfunctional interchange format

While the XMI (XML Metadata Interchange) standard is designed to facilitate the interchange of UML models, it has been largely ineffective in the practical interchange of UML 2.x models. This interoperability ineffectiveness is attributable to two reasons. Firstly, XMI 2.x is large and complex in its own right, since it purports to address a technical problem more ambitious than exchanging UML 2.x models. In particular, it attempts to provide a mechanism for facilitating the exchange of any arbitrary modeling language defined by the OMG's Meta-Object Facility (MOF). Secondly, the UML 2.x Diagram Interchange specification lacks sufficient detail to facilitate reliable interchange of UML 2.x notations between modeling tools. Since UML is a visual modeling language, this shortcoming is substantial for modelers who don't want to redraw their diagrams.

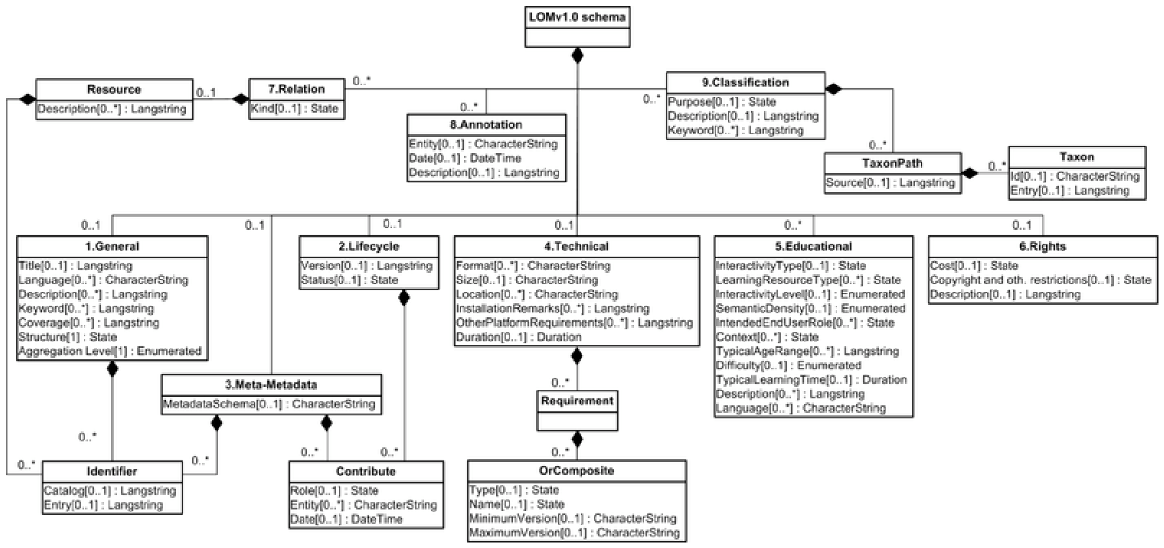
Modeling experts have written sharp criticisms of UML, including Bertrand Meyer's "UML: The Positive Spin", and Brian Henderson-Sellers and Cesar Gonzalez-Perez in "Uses and Abuses of the Stereotype Mechanism in UML 1.x and 2.0".

UML modelling tools

The most well-known UML modelling tool is IBM Rational Rose. Other tools include Rational Rhapsody, MagicDraw UML, StarUML, ArgoUML, Umbrello, BOUML, PowerDesigner, and Dia. Some of popular development environments also offer UML modelling tools, e.g.: Eclipse, NetBeans, and Visual Studio.

Chapter-10

Learning Object Metadata


LOMv1.0 schema
Resource
Description[0..*] : Langstring
7.Relation
Kind[0..1] : State
8.Annotation
Entity[0..1] : CharacterString
Date[0..1] : DateTime
Description[0..1] : Langstring
9.Classification
Purpose[0..1] : State
Description[0..1] : Langstring
Keyword[0..*] : Langstring
TaxonPath
Source[0..1] : Langstring
Taxon
Id[0..1] : CharacterString
Entry[0..1] : Langstring
1.General
Title[0..1] : Langstring
Language[0..*] : CharacterString
Description[0..*] : Langstring
Keyword[0..*] : Langstring
Coverage[0..*] : Langstring
Structure[1] : State
Aggregation Level[1] : Enumerated
2.Lifecycle
Version[0..1] : Langstring
Status[0..1] : State
4.Technical
Format[0..*] : CharacterString
Size[0..1] : CharacterString
Location[0..*] : CharacterString
InstallationRemarks[0..*] : Langstring
OtherPlatformRequirements[0..*] : Langstring
Duration[0..1] : Duration
5.Educational
InteractivityType[0..1] : State
LearningResourceType[0..*] : State
InteractivityLevel[0..1] : Enumerated
SemanticDensity[0..1] : Enumerated
IntendedEndUserRole[0..*] : State
Context[0..*] : State
TypicalAgeRange[0..*] : Langstring
Difficulty[0..1] : Enumerated
TypicalLearningTime[0..1] : Duration
Description[0..*] : Langstring
Language[0..*] : CharacterString
6.Rights
Cost[0..1] : State
Copyright and oth. restrictions[0..1] : State
Description[0..1] : Langstring
3.Meta-Metadata
MetadataSchema[0..1] : CharacterString
Requirement
Identifier
Catalog[0..1] : Langstring
Entry[0..1] : Langstring
Contribute
Role[0..1] : State
Entity[0..*] : CharacterString
Date[0..1] : DateTime
OrComposite
Type[0..1] : State
Name[0..1] : State
MinimumVersion[0..1] : CharacterString
MaximumVersion[0..1] : CharacterString
0..1
0..*

A schematic representation of the hierarchy of elements in the LOM data model

Learning Object Metadata is a data model, usually encoded in XML, used to describe a learning object and similar digital resources used to support learning. The purpose of learning object metadata is to support the reusability of learning objects, to aid discoverability, and to facilitate their interoperability, usually in the context of online learning management systems (LMS).

The IEEE 1484.12.1 – 2002 Standard for Learning Object Metadata is an internationally-recognised open standard (published by the Institute of Electrical and Electronics Engineers Standards Association, New York) for the description of “learning objects”. Relevant attributes of learning objects to be described include: type of object; author; owner; terms of distribution; format; and pedagogical attributes, such as teaching or interaction style.

IEEE 1484.12.1 – 2002 Standard for Learning Object Metadata

In brief

The IEEE working group that developed the standard defined learning objects, *for the purposes of the standard,* as being “any entity, digital or non-digital, that may be used for learning, education or training." This definition has struck many commentators as being rather broad in its scope, but the definition was intended to provide a broad class of objects to which LOM metadata might usefully be associated rather than to give an instructional or pedagogic definition of a learning object. *IEEE 1484.12.1* is the first part of a multipart standard, and describes the LOM data model. The LOM data model specifies which aspects of a learning object should be described and what vocabularies may be used for these descriptions; it also defines how this data model can be amended by additions or constraints. Other parts of the standard are being drafted to define bindings of the LOM data model, i.e. define how LOM records should be represented in XML and RDF (*IEEE 1484.12.3* and *IEEE 1484.12.4* respectively). Here we, focuses on the LOM data model rather than issues relating to XML or other bindings.

IMS Global Learning Consortium is an international consortium that contributed to the drafting of the IEEE Learning Object Metadata (together with the ARIADNE Foundation) and endorsed early drafts of the data model as part of the IMS Learning Resource Meta-data specification (IMS LRM, versions 1.0 – 1.2.2). Feedback and suggestions from the implementers of IMS LRM fed into the further development of the LOM, resulting in some drift between version 1.2 of the IMS LRM specification and what was finally published at the LOM standard. Version 1.3 of the IMS LRM specification realigns the IMS LRM data model with the IEEE LOM data model and specifies that the IEEE XML binding should be used. Thus, we can now use the term 'LOM' in referring to both the IEEE standard and version 1.3 of the IMS specification. The IMS LRM specification also provides an extensive *Best Practice and Implementation Guide*, and an *XSL transform* that can be used to migrate metadata instances from the older versions of the IMS LRM XML binding to the IEEE LOM XML binding.

Technical details

How the data model works

The LOM comprises a **hierarchy of elements**. At the first level, there are nine categories, each of which contains sub-elements; these sub-elements may be simple elements that hold data, or may themselves be aggregate elements, which contain further sub-elements. The semantics of an element are determined by its context: they are affected by the parent or container element in the hierarchy and by other elements in the same container. For example, the various *Description* elements (1.4, 5.10, 6.3, 7.2.2, 8.3 and 9.3) each derive their context from their parent element. In addition, description element 9.3 also takes its context from the value of element 9.1 *Purpose* in the same instance of *Classification*.

The data model specifies that some elements may be repeated either individually or as a group; for example, although the elements 9.3 (*Description*) and 9.1 (*Purpose*) can only occur once within each instance of the *Classification* container element, the *Classification* element may be repeated - thus allowing many descriptions for different purposes.

The data model also specifies the **value space** and **datatype** for each of the simple data elements. The value space defines the restrictions, if any, on the data that can be entered for that element. For many elements, the value space allows any string of Unicode character to be entered, whereas other elements entries must be drawn from a declared list (i.e. a controlled vocabulary) or must be in a specified format (e.g. date and language codes). Some element datatypes simply allow a string of characters to be entered, and others comprise two parts, as described below:

- **LangString** items contain Language and String parts, allowing the same information to be recorded in multiple languages
- **Vocabulary** items are constrained in such a way that their entries have to be chosen from a controlled list of terms - composed of Source-Value pairs - with the Source containing the name of the list of terms being used and the Value containing the chosen term
- **DateTime** and **Duration** items contain one part that allows the date or duration to be given in a machine readable format, and a second that allows a description of the date or duration (for example “mid summer, 1968”).

When implementing the LOM as a data or service provider, it is not necessary to support all the elements in the data model, nor need the LOM data model limit the information which may be provided. The creation of an application profile allows a community of users to specify which elements and vocabularies they will use. Elements from the LOM may be dropped and elements from other metadata schemas may be brought in; likewise, the vocabularies in the LOM may be supplemented with values appropriate to that community.

Requirements

The key requirements for exploiting the LOM as a data or service provider are to:

- Understand user/community needs and to express these as an application profile
- Have a strategy for creating high quality metadata
- Store this metadata in a form which can be exported as LOM records
- Agree a binding for LOM instances when they are exchanged
- Be able to exchange records with other systems either as single instances or *en masse*.

Related specifications

There are many metadata specifications; of particular interest is the Dublin Core Metadata Element Set (commonly known as Simple Dublin Core, standardised as *ANSI/NISO Z39.85 – 2001*), which provides a simpler, more loosely-defined set of elements with some overlap with the LOM, and which is useful for sharing metadata across a wide range of disparate services. The Dublin Core Metadata Initiative is also working on a set of terms which allow the Dublin Core Element Set to be used with greater semantic precision (Qualified Dublin Core). The Dublin Education Working Group aims to provide refinements of Dublin Core for the specific needs of the education community. Details of Dublin Core can be found at the Dublin Core website .

Many other education-related specifications allow for LO metadata to be embedded within XML instances, such as: describing the resources in an IMS Content Package or Resource List; describing the vocabularies and terms in an IMS VDEX (Vocabulary Definition and Exchange) file; and describing the question items in an IMS QTI (Question and Test Interoperability) file. Details of these can be found at the IMS Global website .

The IMS Vocabulary Definition and Exchange (VDEX) specification has a double relation with the LOM, since not only can the LOM provide metadata on the vocabularies in a VDEX instance, but VDEX can be used to describe the controlled vocabularies which are the value space for many LOM elements.

LOM records can be transported between systems using a variety of protocols, perhaps the most widely used being OAI-PMH.

Application profiles

UK LOM Core

For UK Further and Higher Education, the most relevant family of application profiles are those based around the *UK LOM Core* . The UK LOM Core is currently a draft schema researched by a community of practitioners to identify common UK practice in learning object content, by comparing 12 metadata schemas.

CanCore

CanCore provides detailed guidance for the interpretation and implementation of each data element in the LOM standard. These guidelines constitute a 250-page document, and have been developed over three years under the leadership of Norm Friesen, and through consultation with experts across Canada and throughout the world. These guidelines are also available at no charge from the CanCore Website.

ANZ-LOM

ANZ-LOM is a metadata profile developed for the education sector in Australia and New Zealand. The profile provides interpretations of metadata structures and illustrates how to apply controlled vocabularies, especially using the "classification" element. It is supported by detailed examples of learning resource metadata, including regional vocabularies. The ANZ-LOM profile was first published by The Le@rning Federation (TLF) in January, 2008.

Vetadata

The Australian Vocational Training and Education (VET) sector uses an application profile of the IEEE LOM called Vetadata. The profile contains five mandatory elements, and makes use of a number of vocabularies specific to the Australian VET sector. This application profile was first published in 2005. The Vetadata and ANZ-LOM profiles are closely aligned.

NORLOM

NORLOM is the Norwegian LOM profile. The profile is managed by NSSL (The Norwegian Secretariat for Standardization of Learning Technologies)

ISRACore

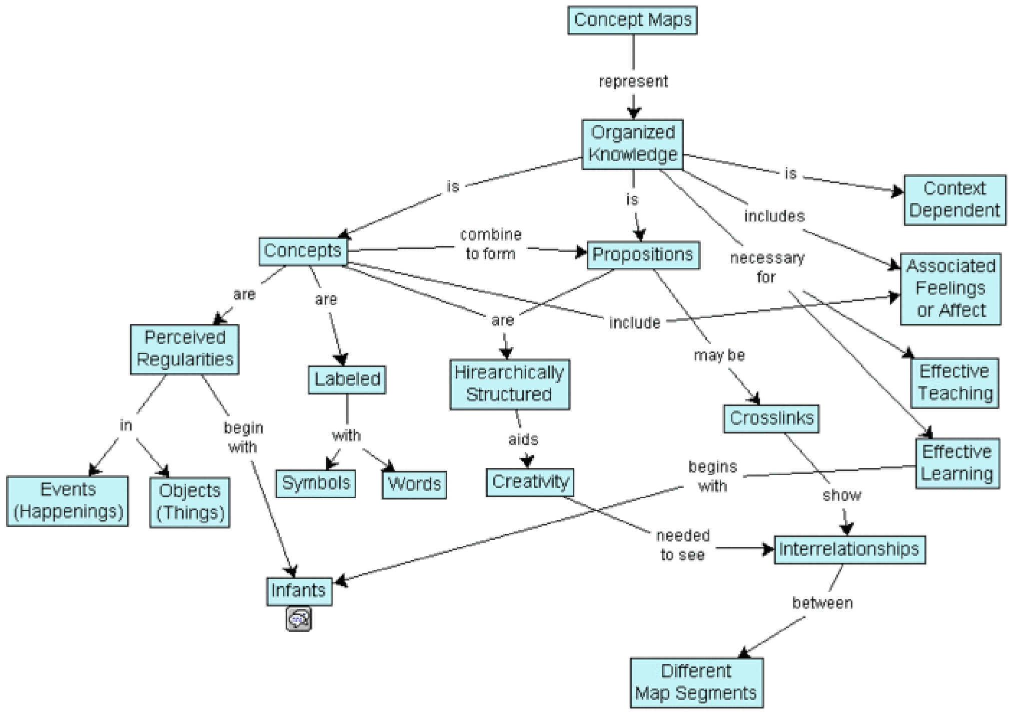
ISRACORE is the Israeli LOM profile. The Israel Internet Association (ISOC-IL) and Inter University Computational Center (IUCC) have teamed up to manage and establish an e-learning objects database.

SWE-LOM

SWE-LOM is the Swedish LOM profile that is managed by IML at Umeå University as a part of the work with the national standardization group TK450 at Swedish Standards Institute.

Chapter-11

Concept Map



Example concept map, created using the IHMC CmapTools computer program.

A **concept map** is a diagram showing the relationships among concepts. It is a graphical tool for organizing and representing knowledge.

Concepts, usually represented as boxes or circles, are connected with labeled arrows in a downward-branching hierarchical structure. The relationship between concepts can be articulated in linking phrases such as "gives rise to", "results in", "is required by," or "contributes to".

The technique for visualizing these relationships among different concepts is called "concept mapping".

An industry standard that implements formal rules for designing at least a subset of such diagrams is the Unified Modeling Language (UML).

Overview

A concept map is a way of representing relationships between ideas, images, or words, in the same way that a sentence diagram represents the grammar of a sentence, a road map represents the locations of highways and towns, and a circuit diagram represents the workings of an electrical appliance. In a concept map, each word or phrase is connected to another and linked back to the original idea, word or phrase. Concept maps are a way to develop logical thinking and study skills, by revealing connections and helping students see how individual ideas form a larger whole.

Concept maps were developed to enhance meaningful learning in the sciences. A well-made concept map grows within a *context frame* defined by an explicit "focus question," while a mind map often has only branches radiating out from a central picture. There is research evidence that knowledge is stored in the brain in the form of productions (situation-response conditionals) that act on declarative memory content which is also referred to as chunks or propositions. Because concept maps are constructed to reflect organization of the declarative memory system, they facilitate sense-making and meaningful learning on the part of individuals who make concept maps and those who use them.

Concept mapping versus topic maps and mind mapping

Concept maps are rather similar to topic maps (in that both allow to connect concepts or topics via graphs), while both can be contrasted with the similar idea of mind mapping, which is often restricted to radial hierarchies and tree structures. Among the various schema and techniques for visualizing ideas, processes, organizations, concept mapping, as developed by Joseph Novak is unique in philosophical basis, which "makes concepts, and propositions composed of concepts, the central elements in the structure of knowledge and construction of meaning." Another contrast between Concept mapping and Mind mapping is the speed and spontaneity when a Mind map is created. A Mind map reflects what you think about a single topic, which can focus group brainstorming. A Concept map can be a map, a system view, of a real (abstract) system or set of concepts. Concept maps are more free form, as multiple hubs and clusters can be created, unlike mind maps which fix on a single conceptual center.

History

The technique of concept mapping was developed by Joseph D. Novak and his research team at Cornell University in the 1970s as a means of representing the emerging science knowledge of students. It has subsequently been used as a tool to increase meaningful learning in the sciences and other subjects as well as to represent the expert knowledge of individuals and teams in education, government and business. Concept maps have their origin in the learning movement called constructivism. In particular, constructivists hold that learners actively construct knowledge.

Novak's work is based on the cognitive theories of David Ausubel (assimilation theory), who stressed the importance of prior knowledge in being able to learn new concepts: "The most important single factor influencing learning is what the learner already knows. Ascertain this and teach accordingly." Novak taught students as young as six years old to make concept maps to represent their response to focus questions such as "What is water?" "What causes the seasons?" In his book *Learning How to Learn*, Novak states that "meaningful learning involves the assimilation of new concepts and propositions into existing cognitive structures."

Various attempts have been made to conceptualize the process of creating concept maps. Ray McAleese, in a series of articles, has suggested that mapping is a process of *off-loading*. In this 1998 paper, McAleese draws on the work of **Sowa** and a paper by **Sweller & Chandler**. In essence, McAleese suggests that the process of making knowledge explicit, using *nodes* and *relationships*, allows the individual to become aware of what they know and as a result to be able to modify what they know. Maria Birbili applies that same idea to helping young children learn to think about what they know. The concept of the **Knowledge Arena** is suggestive of a virtual space where learners may explore what they know and what they do not know.

Use

Concept maps are used to stimulate the generation of ideas, and are believed to aid creativity. For example, concept mapping is sometimes used for brain-storming. Although they are often personalized and idiosyncratic, concept maps can be used to communicate complex ideas.

Formalized concept maps are used in software design, where a common usage is Unified Modeling Language diagramming amongst similar conventions and development methodologies.

Concept mapping can also be seen as a first step in ontology-building, and can also be used flexibly to represent formal argument.

Concept maps are widely used in education and business. Uses include:

- Note taking and summarizing gleaning key concepts, their relationships and hierarchy from documents and source materials
- New knowledge creation: e.g., transforming tacit knowledge into an organizational resource, mapping team knowledge
- Institutional knowledge preservation (retention), e.g., eliciting and mapping expert knowledge of employees prior to retirement

- Collaborative knowledge modeling and the transfer of expert knowledge
- Facilitating the creation of shared vision and shared understanding within a team or organization
- Instructional design: concept maps used as Ausubelian "advance organizers" which provide an initial conceptual frame for subsequent information and learning.
- Training: concept maps used as Ausubelian "advanced organizers" to represent the training context and its relationship to their jobs, to the organization's strategic objectives, to training goals.
- Increasing meaningful learning
- Communicating complex ideas and arguments
- Examining the symmetry of complex ideas and arguments and associated terminology
- Detailing the entire structure of an idea, train of thought, or line of argument (with the specific goal of exposing faults, errors, or gaps in one's own reasoning) for the scrutiny of others.
- Enhancing metacognition (learning to learn, and thinking about knowledge)
- Improving language ability
- Knowledge Elicitation
- Assessing learner understanding of learning objectives, concepts, and the relationship among those concepts
- Lexicon development

Chapter-12

Library Classification

A **library classification** is a system of coding and organizing library materials (books, serials, audiovisual materials, computer files, maps, manuscripts, realia) according to their subject and allocating a call number to that information resource. Similar to classification systems used in biology, bibliographic classification systems group entities together that are similar, typically arranged in a hierarchical tree structure. A different kind of classification system, called a faceted classification system, is also widely used which allows the assignment of multiple classifications to an object, enabling the classifications to be ordered in multiple ways. Books are place on library shelves according to a classification scheme. A basic familiarity with those systems is vital for the student so they can find materials efficiently within the collection. Students of The Master's Seminary will encounter the two main systems: (1) The Library of Congress Classification System [LC], and (2) The Dewey Decimal System [Dewey]. The seminary utilizes the LC system while The Master's College utilizes the Dewey system. The Dewey Decimal System is designed for the library user and browser. It has a logic flow, and once the ten basic categories are memorized, a student can generally find what they are looking for. The LC system is not designed with the library patron in mind. The LC system is really designed for a "closed stacks" library where patrons do not "browse" the collection but rather hand a page a request for a book, who would then retrieve it for the patron. As a result the LC system is rather notorious for strange and often logic defying "call num-bers."1 We have listed the LC and Dewey systems both in total and then more detailed sections for Biblical Studies and Theology. We have also added a section here to explain the call numbers for Biblical Commentaries in the LC scheme. There are a couple of other classification systems in use including, The Bliss Classification System, the Colon Classification System, the more widely used Universal Decimal Classification System. For systems particular to theological libraries the Union Seminary Classification System is the most well known (formerly used by the Grace Theological Seminary library), although it is now little used and will be only be encountered by students who happen into an archive or part of an unconverted collection. There was a Westminster Seminary Classification System, (based on the Union system) but that system is now entirely de-funct, Westminster Seminary eliminated it itself more than a decade ago and we are unaware of any institution that currently uses it at any level. The question is always asked about a personal system for office and files. Keeping track of ones own library and personal files is obviously important. In this course we will be recommending that students invest in a computer program called Pro-Cite, which is a personal bibliographic database. This system will allow for search based on key word or any other criteria. It also allows for direct interface through the Internet to other library systems so that they can be search and records imported. Personally, we recommend that individuals, especially pastors, use the Dewey system as a basis for their own filing system. It is easy to use, if you enlist the aid of others in either setting up or maintaining the system; most people at least have a working familiarity with Dewey. For those who wish a more elaborate system, Dr. Rod-ney Decker of Baptist Bible College and Seminary, has put together a quite detailed system, which he recently updated to a third edition. The most important aspect of a personal system is that it be functional; that is, it can handle a library up to per-haps 5,000 volumes, and that the categories can be expanded as necessary. It should also be friendly; that is, 1 The "Call Number" is the number of the classification system (Dewey or LC) assigned to that particular book. No two books in a collection will (when they are properly catalogued) have the same "Call Number." Biblical and Theological Studies in the LC system can be particularly maddening. For instance, works on warfare in the Biblical era might be found in either the BR section in "Bible Backgrounds"; or it might be found in the DS sec-tion for the "History of Israel" or in the little-used U Section for "Military Science." There is often no particu-lar rhyme or reason. Of particular note to students are Greek and Hebrew reference and word study works. They may either be located in the BS section of the PA section. Students should check both locations. you don't have to spend hours explaining it to someone (like your secretary) on how it works. It should also be feasible; that is, it shouldn't require the investment of months of your life to create and implement.

Description

Library classification forms part of the field of library and information science. It is a form of bibliographic classification (library classifications are used in library catalogs, while "bibliographic classification" also covers classification used in other kinds of bibliographic databases). It goes hand in hand with library (descriptive) cataloging under the rubric of *cataloging and classification*, sometimes grouped together as *technical services*. The library professional who engages in the process of cataloging and classifying library materials is called a *cataloguer* or *catalog librarian*. Library classification systems are one of the two tools used to facilitate subject access. The other consists of alphabetical indexing languages such as Thesauri and Subject Headings systems.

Library classification of a piece of work consists of two steps. Firstly the "aboutness" of the material is ascertained. Next, a call number (essentially a book's address), based on the classification system in use at the particular library will be assigned to the work using the notation of the system.

It is important to note that unlike subject heading or thesauri where multiple terms can be assigned to the same work, in library classification systems, each work can only be placed in one class. This is due to shelving purposes: A book can have only one physical place. However in classified catalogs one may have main entries as well as added entries.

Most classification systems like the Dewey Decimal Classification (DDC) and Library of Congress classification also add a cutter number to each work which adds a code for the author of the work.

Classification systems in libraries generally play two roles. Firstly they facilitate subject access by allowing the user to find out what works or documents the library has on a certain subject. Secondly, they provide a known location for the information source to be located (e.g. where it is shelved).

Until the 19th century, most libraries had closed stacks, so the library classification only served to organize the subject catalog. In the 20th century, libraries opened their stacks to the public and started to shelve the library material itself according to some library classification to simplify subject browsing.

Some classification systems are more suitable for aiding subject access, rather than for shelf location. For example, UDC which uses a complicated notation including plus, colons are more difficult to use for the purpose of shelf arrangement but are more expressive compared to DDC in terms of showing relationships between subjects. Similarly faceted classification schemes are more difficult to use for shelf arrangement, unless the user has knowledge of the citation order.

Depending on the size of the library collection, some libraries might use classification systems solely for one purpose or the other. In extreme cases a public library with a small collection might just use a classification system for location of resources but might not use a complicated subject classification system. Instead all resources might just be put into a couple of wide classes (Travel, Crime, Magazines etc.). This is known as a "mark and park" classification method, more formally called reader interest classification.

Types

There are many standard system of library classification in use, and many more have been proposed over the years. However in general, Classification systems can be divided into three types depending on how they are used.

- Universal schemes covering all subjects. Examples include Dewey Decimal Classification, Universal Decimal Classification and Library of Congress Classification

- Specific classification schemes for particular subjects or types of materials. Examples include Iconclass, British Catalogue of Music Classification, and Dickinson classification, or the NLM Classification for medicine.

- National schemes specially created for certain countries. An example is the Swedish library classification system, SAB (Sveriges Allmänna Biblioteksförening).

In terms of functionality, classification systems are often described as

- enumerative: produce an alphabetical list of subject headings, assign numbers to each heading in alphabetical order

library classification is the technical process

- hierarchical: divides subjects hierarchically, from most general to most specific
- faceted or analytico-synthetic: divides subjects into mutually exclusive orthogonal facets

There are few completely enumerative systems or faceted systems, most systems are a blend but favouring one type or the other. The most common classification systems, LCC and DDC, are essentially enumerative, though with some hierarchical and faceted elements (more so for DDC), especially at the broadest and most general level. The first true faceted system was the Colon classification of S. R. Ranganathan.

Universal classification systems used in the English-speaking world

- Bliss bibliographic classification (BC)
- Dewey Decimal Classification (DDC)
- Library of Congress Classification (LCC)

(The above systems are the most common in the English-speaking world.)

- BISAC Subject Headings: The publishing industry standard for classification that is being adopted by some libraries.
- Harvard-Yenching Classification: An English classification system for Chinese language materials.
- V-LIB 1.2 (2008 Vartavan Library Classification for over 700 fields of knowledge, currently sold under license in the UK by Rosecastle Ltd.

Universal classification systems in other languages

- A system of book classification for Chinese libraries (Liu's Classification) library classification for user
    - New Classification Scheme for Chinese Libraries
- Nippon Decimal Classification (NDC)
- Chinese Library Classification (CLC)
- Korean Decimal Classification (KDC)
- Library-Bibliographic Classification (BBK) from Russia.

Universal classification systems that rely on synthesis (faceted systems)

- Bliss bibliographic classification

- Colon classification
- Cutter Expansive Classification
- Universal Decimal Classification

Newer classification systems tend to use the principle of synthesis (combining codes from different lists to represent the different attributes of a work) heavily, which is comparatively lacking in LC or DDC.

Comparing Classification Systems

As a result of differences in Notation, history, use of enumeration, hierarchy, facets, classification systems can differ in the following ways

- Type of Notation: Notation can be pure (consisting of only numerals for example) or mixed (consisting of letters, numerals, and other symbols).

- Expressiveness: This is the degree in which the notation can express relationship between concepts or structure.

- Whether they support mnemonics: For example the number 44 in DDC notation usually means it concerns some aspect of France. For example 598.0944 concerns "Birds in France". the 09 signifies country code, and 44 represents France.

- Hospitality: The degree in which the system is able to accommodate new subjects

- Brevity: Length of the notation to express the same concept

- Speed of updates and degree of support: The best classification systems are constantly being reviewed and improved.

- Consistency

- Simplicity

- Usability

Chapter-13

Linear Belief Function

Linear Belief Function is an extension of the Dempster-Shafer theory of belief functions to the case when variables of interest are continuous. Examples of such variables include financial asset prices, portfolio performance, and other antecedent and consequent variables. The theory was originally proposed by Arthur P. Dempster in the context of Kalman Filters and later was reelaborated, refined, and applied to knowledge representation in artificial intelligence and decision making in finance and accounting by Liping Liu .

Concept

A linear belief function intends to represent our belief regarding the location of the true value as follows: We are certain that the truth is on a so-called certainty hyperplane but we do not know its exact location; along some dimensions of the certainty hyperplane, we believe the true value could be anywhere from $-\infty$ to $+\infty$ and the probability of being at a particular location is described by a normal distribution; along other dimensions, our knowledge is vacuous, i.e., the true value is somewhere from $-\infty$ to $+\infty$ but the associated probability is unknown. As we know, a belief function in general is defined by a mass function over a class of focal elements, which may have nonempty intersections. A linear belief function is a special type of belief functions in the sense that its focal elements are exclusive, parallel sub-hyperplanes over the certainty hyperplane and its mass function is a normal distribution across the sub-hyperplanes.

Based on the above geometrical description, Shafer and Liu propose two mathematical representations of a LBF: a wide-sense inner product and a linear functional in the variable space, and as their duals over a hyperplane in the sample space. Monney proposes a still another structure called Gaussian hints. Although these representations are mathematically neat, they tend to be unsuitable for knowledge representation in expert systems.

Knowledge Representation

A linear belief function can represent both logical and probabilistic knowledge for three types of variables: deterministic such as an observable or controllable, random whose distribution is normal, and vacuous on which no knowledge bears. Logical knowledge is represented by linear equations, or geometrically, a certainty hyperplane. Probabilistic knowledge is represented by a normal distribution across all parallel focal elements.

In general, assume X is a vector of multiple normal variables with mean μ and covariance Σ. Then, the multivariate normal distribution can be equivalently represented as a moment matrix:

$$M(X) = \left(\begin{array}{c} \mu \\ \Sigma \end{array} \right).$$

If the distribution is non-degenerate, i.e., Σ has a full rank and its inverse exists, the moment matrix can be fully swept:

$$M(\vec{X}) = \left(\begin{array}{c} \mu\Sigma^{-1} \\ -\Sigma^{-1} \end{array} \right)$$

Except for normalization constant, the above equation completely determines the normal density function for *X*. Therefore, $M(\vec{X})$ represents the probability distribution of *X* in the potential form.

These two simple matrices allow us to represent three special cases of linear belief functions. First, for an ordinary normal probability distribution M(X) represents it. Second, suppose one makes a direct observation on X and obtains a value μ. In this case, since there is no uncertainty, both variance and covariance vanish, i.e., Σ = 0. Thus, a direct observation can be represented as:

$$M(X) = \left(\begin{array}{c} \mu \\ 0 \end{array} \right)$$

Third, suppose one is completely ignorant about X. This is a very thorny case in Bayesian statistics since the density function does not exist. By using the fully swept moment matrix, we represent the vacuous linear belief functions as a zero matrix in the swept form follows:

$$M(\vec{X}) = \left[\begin{array}{c} 0 \\ 0 \end{array} \right]$$

One way to understand the representation is to imagine complete ignorance as the limiting case when the variance of X approaches to ∞, where one can show that $\Sigma^{-1} = 0$ and hence $M(\vec{X})$ vanishes. However, the above equation is not the same as an improper prior or normal distribution with infinite variance. In fact, it does not correspond to any unique probability distribution. For this reason, a better way is to understand the vacuous linear belief functions as the neutral element for combination.

To represent the remaining three special cases, we need the concept of partial sweeping. Unlike a full sweeping, a partial sweeping is a transformation on a subset of variables. Suppose X and Y are two vectors of normal variables with the joint moment matrix:

$$M(X,Y) = \begin{bmatrix} \mu_1 & \mu_2 \\ \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

Then M(X, Y) may be partially swept. For example, we can define the partial sweeping on X as follows:

$$M(\vec{X},Y) = \begin{bmatrix} \mu_1(\Sigma_{11})^{-1} & \mu_2 - \mu_1(\Sigma_{11})^{-1}\Sigma_{12} \\ -(\Sigma_{11})^{-1} & (\Sigma_{11})^{-1}\Sigma_{12} \\ \Sigma_{21}(\Sigma_{11})^{-1} & \Sigma_{22} - \Sigma_{21}(\Sigma_{11})^{-1}\Sigma_{12} \end{bmatrix}$$

If X is one-dimensional, a partial sweeping replaces the variance of X by its negative inverse and multiplies the inverse with other elements. If X is multidimensional, the operation involves the inverse of the covariance matrix of X and other multiplications. A swept matrix obtained from a partial sweeping on a subset of variables can be equivalently obtained by a sequence of partial sweepings on each individual variable in the subset and the order of the sequence does not matter. Similarly, a fully swept matrix is the result of partial sweepings on all variables.

We can make two observations. First, after the partial sweeping on X, the mean vector and covariance matrix of X are respectively $\mu_1(\Sigma_{11})^{-1}$ and $-(\Sigma_{11})^{-1}$, which are the same as that of a full sweeping of the marginal moment matrix of X. Thus, the elements corresponding to X in the above partial sweeping equation represent the marginal distribution of X in potential form. Second, according to statistics, $\mu_2 - \mu_1(\Sigma_{11})^{-1}\Sigma_{12}$is the conditional mean of Y given X = 0; $\Sigma_{22} - \Sigma_{21}(\Sigma_{11})^{-1}\Sigma_{12}$ is the conditional covariance matrix of Y given X = 0; and $(\Sigma_{11})^{-1}\Sigma_{12}$ is the slope of the regression model of Y on X. Therefore, the elements corresponding to Y indices and the intersection of X and Y in $M(\vec{X},Y)$represents the conditional distribution of Y given X = 0.

These semantics render the partial sweeping operation a useful method for manipulating multivariate normal distributions. They also form the basis of the moment matrix representations for the three remaining important cases of linear belief functions, including proper belief functions, linear equations, and linear regression models.

Proper Linear Belief Functions

For variables X and Y, assume there exists a piece of evidence justifying a normal distribution for variables Y while bearing no opinions for variables X. Also, assume that X and Y are not perfectly linearly related, i.e., their correlation is less than 1. This case involves a mix of an ordinary normal distribution for Y and a vacuous belief function for X. Thus, we represent it using a partially swept matrix as follows:

$$M(\vec{X},Y) = \begin{bmatrix} 0 & \mu_2 \\ 0 & 0 \\ 0 & \Sigma_{22} \end{bmatrix}$$

This is how we could understand the representation. Since we are ignorant on X, we use its swept form and set $\mu_1(\Sigma_{11})^{-1} = 0$ and $-(\Sigma_{11})^{-1} = 0$. Since the correlation between X and Y is less than 1, the regression coefficient of X on Y approaches to 0 when the variance of X approaches to ∞. Therefore, $(\Sigma_{11})^{-1}\Sigma_{12} = 0$. Similarly, one can prove that $\mu_1(\Sigma_{11})^{-1}\Sigma_{12} = 0$ and $\Sigma_{21}(\Sigma_{11})^{-1}\Sigma_{12} = 0$.

Linear Equations

Suppose X and Y are two row vectors, and Y = XA + b, where A and b are the coefficient matrices. We represent the equation using a partially swept matrix as follows:

$$M(\vec{X},Y) = \begin{bmatrix} 0 & b \\ 0 & A \\ A^T & 0 \end{bmatrix}$$

We can understand the representation based on the fact that a linear equation contains two pieces of knowledge: (1) complete ignorance about all variables; and (2) a degenerate conditional distribution of dependent variables given independent variables. Since X is an independent vector in the equation, we are completely ignorant about it. Thus, $\mu_1(\Sigma_{11})^{-1} = 0$ and $-(\Sigma_{11})^{-1} = 0$. Given X = 0, Y is completely determined to be b. Thus, the conditional mean of Y is b and the conditional variance is 0. Also, the regression coefficient matrix is A.

Note that the knowledge to be represented in linear equations is very close to that in a proper linear belief functions, except that the former assumes a perfect correlation between X and Y while the latter does not. This observation is interesting; it characterizes the difference between partial ignorance and linear equations in one parameter — correlation.

Linear Regression Models

A linear regression model is a more general and interesting case than previous ones. Suppose X and Y are two vectors and Y = XA + b + E, where A and b are the appropriate coefficient matrices and E is an independent white noise satisfying E ~ N(0, Σ). We represent the model as the following partially swept matrix:

$$M(\vec{X}, Y) = \begin{bmatrix} 0 & b \\ 0 & A \\ A^T & \Sigma \end{bmatrix}$$

This linear regression model may be considered as the combination of two pieces of knowledge, one is specified by the linear equation involving three variables X, Y, and E, and the other is a simple normal distribution of E, i.e., E ~ N(0, Σ). Alternatively, one may consider it similar to a linear equation, except that, given X = 0, Y is not completely determined to be b. Instead, the conditional mean of Y is b while the conditional variance is Σ. Note that, in this alternative interpretation, a linear regression model forms a basic building block for knowledge representation and is encoded as one moment matrix. Besides, the noise term E does not appear in the representation. Therefore, it makes the representation more efficient.

From representing the six special cases, we see a clear advantage of the moment matrix representation, i.e., it allows a unified representation for seemingly diverse types of knowledge, including linear equations, joint and conditional distributions, and ignorance. The unification is significant not only for knowledge representation in artificial intelligence but also for statistical analysis and engineering computation. For example, the representation treats the typical logical and probabilistic components in statistics — observations, distributions, improper priors (for Bayesian statistics), and linear equation models — not as separate concepts, but as manifestations of a single concept. It allows one to see the inner connections between these concepts or manifestations and to interplay them for computational purposes.

Knowledge Operations

There are two basic operations for making inferences in expert systems using linear belief functions: combination and marginalization. Combination corresponds to the integration of knowledge whereas marginalization corresponds to the coarsening of knowledge. Making an inference involves combining relevant knowledge into a full body of knowledge and then projecting the full body of knowledge to a partial domain, in which an inference question is to be answered.

Marginalization

Marginalization projects a linear belief function into one with fewer variables. Expressed as a moment matrix, it is simply the restriction of a nonswept moment matrix to a submatrix corresponding to the remaining variables. For example, for the joint distribution M(X, Y), its marginal to Y is:

$$M^{\downarrow Y}(X,Y) = \left[\begin{array}{c} \mu_2 \\ \Sigma_{22} \end{array} \right]$$

When removing a variable, it is important that the variable has not been swept on in the corresponding moment matrix, i.e., it does not have an arrow sign above the variable. For example, projecting the matrix $M(\vec{X},Y)$ to Y produces:

$$M^{\downarrow Y}(\vec{X},Y) = \left[\begin{array}{c} \mu_2 - \mu_1(\Sigma_{11})^{-1}\Sigma_{12} \\ \Sigma_{22} - \Sigma_{21}(\Sigma_{11})^{-1}\Sigma_{12} \end{array} \right]$$

which is not the same linear belief function of Y. However, it is easy to see that removing any or all variables in Y from the partially swept matrix will still produce the correct result — a matrix representing the same function for the remaining variables.

To remove a variable that has been already swept on, we have to reverse the sweeping using partial or full reverse sweepings. Assume $M(\vec{X})$ is a fully swept moment matrix,

$$M(\vec{X}) = \left(\begin{array}{c} \bar{\mu} \\ \bar{\Sigma} \end{array} \right)$$

Then a full reverse sweeping of $M(\vec{X})$ will recover the moment matrix M(X) as follows:

$$M(X) = \left(\begin{array}{c} -\bar{\mu}\bar{\Sigma}^{-1} \\ -\bar{\Sigma}^{-1} \end{array} \right)$$

If a moment matrix is in a partially swept form, say

$$M(\vec{X},Y) = \left[\begin{array}{cc} \bar{\mu}_1 & \bar{\mu}_2 \\ \bar{\Sigma}_{11} & \bar{\Sigma}_{12} \\ \bar{\Sigma}_{21} & \bar{\Sigma}_{22} \end{array} \right]$$

its partially reverse sweeping on X is defined as follows:

$$M(X,Y) = \left[\begin{array}{cc} -\bar{\mu}_1(\bar{\Sigma}_{11})^{-1} & \bar{\mu}_2 - \bar{\mu}_1(\bar{\Sigma}_{11})^{-1}\bar{\Sigma}_{12} \\ -(\bar{\Sigma}_{11})^{-1} & -(\bar{\Sigma}_{11})^{-1}\bar{\Sigma}_{12} \\ -\bar{\Sigma}_{21}(\bar{\Sigma}_{11})^{-1} & \bar{\Sigma}_{22} - \bar{\Sigma}_{21}(\bar{\Sigma}_{11})^{-1}\bar{\Sigma}_{12} \end{array} \right]$$

Reverse sweepings are similar to those of forward ones, except for a sign difference for some multiplications. However, forward and reverse sweepings are opposite operations. It can be easily shown that applying the fully reverse sweeping to $M(\vec{X})$ will recover the initial moment matrix M(X). It can also be proved that applying a partial reverse sweeping on X to the matrix $M(\vec{X}, Y)$ will recover the moment matrix M(X,Y). As a matter of fact, Liu proves that a moment matrix will be recovered through a reverse sweeping after a forward sweeping on the same set of variables. It can be also recovered through a forward sweeping after a reverse sweeping. Intuitively, a partial forward sweeping factorizes a joint into a marginal and a conditional, whereas a partial reverse sweeping multiplies them into a joint.

Combination

According to Dempster's rule, the combination of belief functions may be expressed as the intersection of focal elements and the multiplication of probability density functions. Liping Liu applies the rule to linear belief functions in particular and obtains a formula of combination in terms of density functions. Later he proves a claim by Arthur P. Dempster and reexpresses the formula as the sum of two fully swept matrices. Mathematically, assume $M_1(\vec{X}) = \begin{pmatrix} \bar{\mu}_1 \\ \bar{\Sigma}_1 \end{pmatrix}$ and $M_2(\vec{X}) = \begin{pmatrix} \bar{\mu}_2 \\ \bar{\Sigma}_2 \end{pmatrix}$ are two LBFs for the same vector of variables X. Then their combination is a fully swept matrix:

$$M(\vec{X}) = \begin{pmatrix} \bar{\mu}_1 + \bar{\mu}_2 \\ \bar{\Sigma}_1 + \bar{\Sigma}_2 \end{pmatrix}$$

This above equation is often used for multiplying two normal distributions. Here we use it to define the combination of two linear belief functions, which include normal distributions as a special case. Also, note that a vacuous linear belief function (0 swept matrix) is the neutral element for combination. When applying the equation, we need to consider two special cases. First, if two matrices to be combined have different dimensions, then one or both matrices must be vacuously extended, i.e., assuming ignorance on the variables that are no present in each matrix. For example, if M_1(X,Y) and M_2(X,Z) are to be combined, we will first extend them into $M_1(X, Y, \vec{Z})$ and $M_2(X, \vec{Y}, Z)$ respectively such that $M_1(X, Y, \vec{Z})$ is ignorant about Z and $M_2(X, \vec{Y}, Z)$ is ignorant about Y. The vacuous extension was initially proposed by Kong for discrete belief functions. Second, if a variable has zero variance, it will not permit a sweeping operation. In this case, we can pretend the variance to be an extremely small number, say ε, and perform the desired sweeping and combination. We can then apply a reverse sweeping to the combined matrix on the same variable and let ε approaches 0. Since zero variance means complete certainty about a variable, this ε-procedure will vanish ε terms in the final result.

In general, to combine two linear belief functions, their moment matrices must be fully swept. However, one may combine a fully swept matrix with a partially swept one directly if the variables of the former matrix have been all swept on in the later. We can use the linear regression model — Y = XA + b + E — to illustrate the property. As we mentioned, the regression model may be considered as the combination of two pieces of knowledge: one is specified by the linear equation involving three variables X, Y, and E, and the other is a simple normal distribution of E, i.e., E ~ N(0, Σ). Let

$M_1(\vec{X}, \vec{\mathrm{E}}, Y) = \begin{bmatrix} 0 & 0 & b \\ 0 & 0 & A \\ 0 & 0 & I \\ A^T & I & 0 \end{bmatrix}$ and $M_2(\vec{\mathrm{E}}) = \begin{bmatrix} 0 \\ -\Sigma^{-1} \end{bmatrix}$ be their moment matrices respectively. Then the two matrices can be combined directly without sweeping $M_1(\vec{X}, \vec{\mathrm{E}}, Y)$ on Y first. The result of the combination is a partially swept matrix as follows:

$$M(\vec{X}, \vec{\mathrm{E}}, Y) = \begin{bmatrix} 0 & 0 & b \\ 0 & 0 & A \\ 0 & -\Sigma^{-1} & I \\ A^T & I & 0 \end{bmatrix}$$

If we apply a reverse sweeping on E and then remove E from the matrix, we will obtain the same representation of the regression model.

Applications

We may use an audit problem to illustrate the three types of variables as follows. Suppose we want to audit the ending balance of accounts receivable (E). As we saw earlier, E is equal to the beginning balance (*B*) plus the sales (S) for the period minus the cash receipts (*C*) on the sales plus a residual (*R*) that represents insignificant sales returns and cash discounts. Thus, we can represent the logical relation as a linear equation:

$$E = B + S - C + R$$

Furthermore, if the auditor believes E and B are 100 thousand dollars on the average with a standard deviation 5 and the covariance 15, we can represent the belief as a multivariate normal distribution. If historical data indicate that the residual R is zero on the average with a standard deviation of 0.5 thousand dollars, we can summarize the historical data by normal distribution R ~ N(0, 0.5^2). If there is a direct observation on cash receipts, we can represent the evidence as an equation say, C = 50 (thousand dollars). If the auditor knows nothing about the beginning balance of accounts receivable, we can represent his or her ignorance by a vacuous LBF. Finally, if historical data suggests that, given cash receipts C, the sales S is on the average 8C + 4 and has a standard deviation 4 thousand dollars, we can represent the knowledge as a linear regression model S ~ N(4 + 8C, 16).

Chapter-14

Logico-linguistic Modeling

Logico-linguistic modeling is a method for building knowledge-based systems with a learning capability using Conceptual Models from Soft systems methodology, modal predicate logic and the Prolog artificial intelligence language.

Overview

Logico-linguistic modeling is a six stage method developed primarily for building knowledge-based systems (KBS), but it also has application in manual decision support systems and information source analysis. Logico-linguistic models have a superficial similarity to Sowa's Conceptual Graphs, both use bubble style diagrams, both are concerned with concepts, both can be expressed in logic and both can be used in artificial intelligence. However, logico-linguistic models are very different in both logical form and in their method of construction.

Logico-linguistic modeling was developed in order to solve theoretical problems found in the Soft Systems method for information system design. The main thrust of the research into has been to show how Soft Systems Methodology (SSM), a method of systems analysis, can be extended into artificial intelligence.

Background

SSM employs three modeling devices i.e. rich pictures, root definitions, and Conceptual Models of human activity systems. The root definitions and conceptual models are built by stakeholders themselves in an iterative debate organized by a facilitator. The strengths of this method lie, firstly, in its flexibility, the fact that it can address any problem situation, and, secondly, in the fact that the solution belongs to the people in the organization and is not imposed by an outside analyst.

Information Requirements Analysis (IRA) took the basic SSM method a stage further and showed showed how the Conceptual Models could be developed into a detailed information system design. IRA calls for the addition of two modeling devices: "Information Categories" which show the required information inputs and outputs from the activities identified in an expanded conceptual model; and the "Maltese Cross" a matrix which shows the inputs and outputs from the information categories and shows where new information processing procedures are required. A completed Maltese Cross is sufficient for the detailed design of a transaction processing system.

The initial impetus to the development of logico-linguistic modeling was a concern with the theoretical problem of how an information system can have a connection to the physical world. This is a problem in both IRA and more established methods (such as SSADM) because none base their information system design on models of the physical world. IRA designs are based on a notional conceptual model and SSADM is based on models of the movement of documents.

The solution to these problems provided a formula that was not limited to the design of transaction processing systems but could be used for the design of KBS with learning capability.

The Six Stages of logico-linguistic modeling

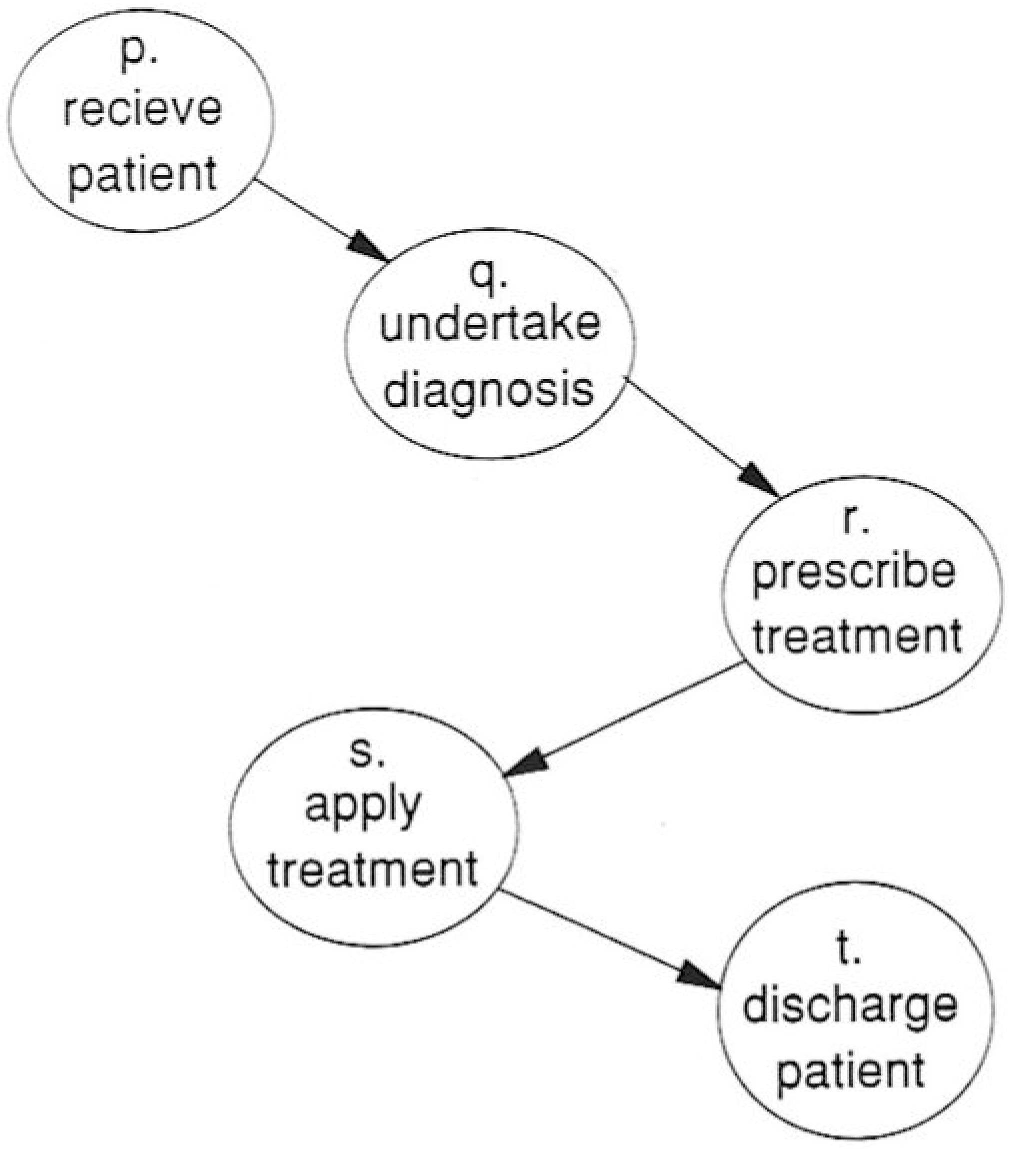


Fig 1. SSM Conceptual Model

The logico-linguistic modeling method comprises six stages.

1. Systems Analysis

In the first stage logico-linguistic modeling uses SSM for systems analysis. This stage seeks to structure the problem in the client organization by identifying stakeholders, modelling organizational objectives and discussing possible solutions. At this stage it not assumed that a KBS will be a solution and logico-linguistic modeling often produces solutions that do not require a computerized KBS.

Expert systems tend to capture the expertise, of individuals in different organizations, on the same topic. By contrast a KBS, produced by logico-linguistic modeling, seeks to capture the expertise of individuals in the same organization on different topics. The emphasis is on the elicitation of organizational or group knowledge rather than individual experts. In logico-linguistic modeling the stakeholders become the experts.

The end point of this stage is an SSM style conceptual models such as figure 1.

2. Language Creation

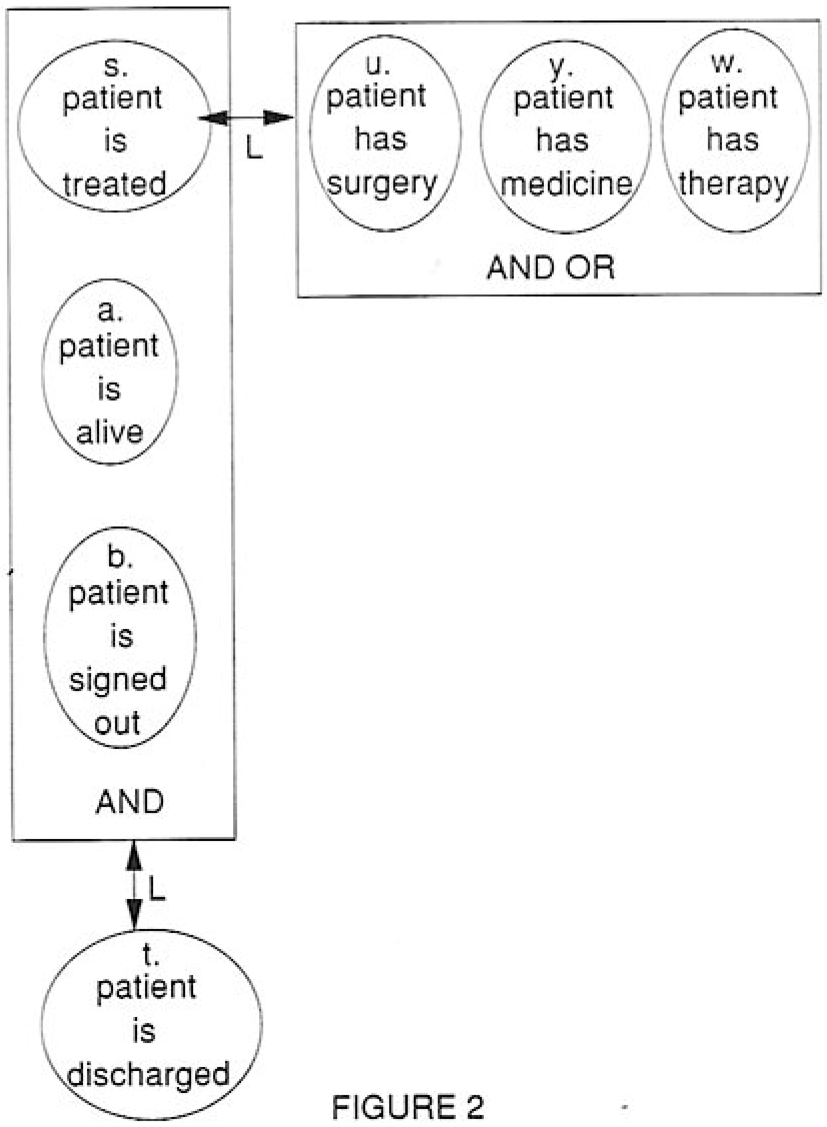


FIGURE 2

Fig 2. Logico-linguistic Model

According to the theory behind logico-linguistic modeling the SSM conceptual model building process is a Wittgensteinian language-game in which the stakeholders build a language to describe the problem situation. The logico-linguistic model expresses this language as a set of definitions, see figure 2.

3. Knowledge Elicitation

After the model of the language has been built putative knowledge about the real world can be added by the stakeholders. Traditional SSM conceptual models contain only one logical connective (a necessary condition). In order to represent causal sequences, “sufficient condition” and “necessary & sufficient conditions” are also required. In logico-linguistic modeling this deficiency is remedied by two addition types of connective. The outcome of stage three is an empirical model, see figure 3.

4. Knowledge Representation

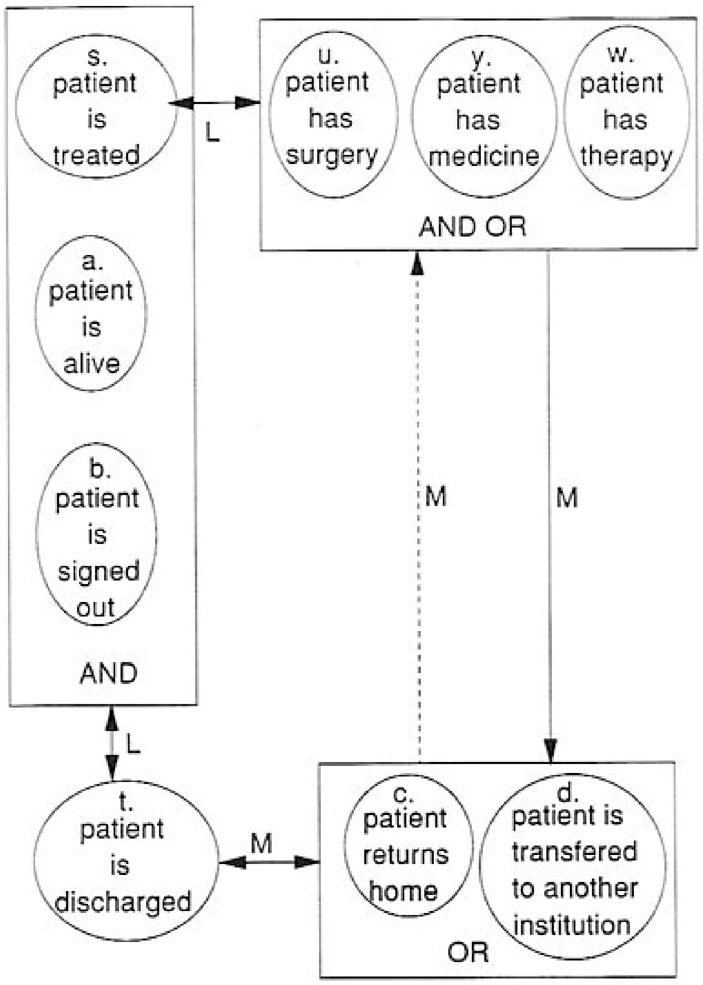


FIGURE 3

Fig 3. Empirical Model

Modal predicate logic (a combination of modal logic and predicate logic) is used as the formal method of knowledge representation. The connectives from the language model are logically true (indicated by the "*L*" modal operator) and connective added at the knowledge elicitation stage are possibility true (indicated by the "*M*" modal operator). Before proceeding to stage 5, the models are expressed in logical formulae.

5. Computer code

Formulae in predicate logic translate easily into the Prolog artificial intelligence language. The modality is expressed by two different types of Prolog rules. Rules taken from the language creation stage of model building process are treated as incorrigible. While rules from the knowledge elicitation stage are marked as hypothetical rules. The system is not confined to decision support but has a built in learning capability.

6. Verification

A knowledge based system built using this method verifies itself. Verification takes place when the KBS is used by the clients. It is an on going process that continues throughout the life of the system. If the stakeholder beliefs about the real world are mistaken this will be brought out by the addition of Prolog facts that conflict with the hypothetical rules. It operates in accordance to the classic principle of falsifiability found in the philosophy of science

Applications

- **Knowledge-based computer systems**

Logico-linguistic modeling has been used to produce fully operational computerized knowledge based systems, such as one for the management of diabetes patients in a hospital out-patients department.

- **Manual decision support**

In other projects the need to move into Prolog was considered unnecessary because the printed logico-linguistic models provided an easy to use guide to decision making. For example, a system for mortgage loan approval

- **Information source analysis**

In some cases a KBS could not be built because the organization did not have all the knowledge needed to support all their activities. In these cases logico-linguistic modeling showed shortcomings in the supply of information and where more was needed. For example, a planning department in a telecoms company

Criticism

While logico-linguistic Modeling overcomes the problems, found in SSM, in the transition from conceptual model to computer code, it does so at the expense of making the stakeholder constructed models much more complex. It has been argued that the benefits of this complexity is questionable and that this modeling method is much harder to use than other methods

This contention has been born out by subsequent research. In an attempt to use logico-linguistic modeling to model buying decisions across twelve companies, the researcher needed to simplify the models and the modal elements were discarded.