Paper

Structural representations of unstructured knowledge

Wiesław Traczyk

Abstract—Knowledge should be represented in a formal, structured manner if we want to process and manage it. Unfortunately a source knowledge presented in many documents has informal, unstructured shape. The goal of these considerations is to present the methods of translation from the textual, unstructured knowledge to the structured knowledge, preserving textual form.

Keywords*— *textual knowledge, knowledge representation languages, ontology.

1. Introduction

Knowledge is used in all areas of human activities but there are domains, relevant to computer applications, in which the word *knowledge* is located in the name of a branch. Three more popular such fields, with their links to data, knowledge and human agents, are presented in Fig. 1.

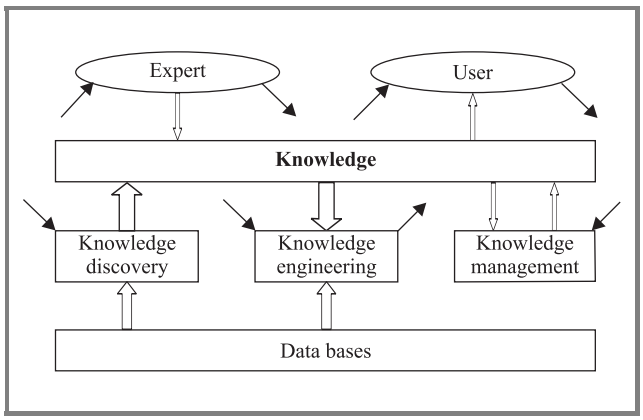


Fig. 1. Knowledge generation, processing and utilization.

Knowledge discovery looks for specific, previously unknown but important, regularities in large data bases. These regularities (or patterns) determine a new knowledge gained from data. Different forms of knowledge obtained from data mining are adapted to the goals and dictated by experts. Usually patterns are described by association rules, decision trees, regression functions and neural networks.

Knowledge engineering constructs knowledge-based systems used for reasoning, intelligent search and decision support. Knowledge, related to particular domain, is built into a system and utilized for data transformation (e.g., from *conditions* to *conclusions*).

Knowledge management consists in organization and facilitation of knowledge generation and utilization, with a main task – increase of institution profit.

In the last two domains a form of primary knowledge can be very diversified. The low and company regulations, institution rules, medical procedures, web pages, fragments of books and other documents written in a natural language are frequently the main sources of knowledge. But, if the knowledge is to be processed by computers, its form has to be converted to a formal shape, with precisely defined syntax and semantics. Then, the proper *knowledge representation* (KR) is needed, easy to obtain from natural language descriptions and easy to understand by computers. General demands for such a representation are usually summarized by *the ontology*: a set of formally specified concepts (such as things or events) and their properties and relations that describe a domain of interest, in order to create an agreed-upon vocabulary for exchanging information.

Since the goal of KR is to express knowledge in computer-tractable form, the two questions should be answered before discussing the possible solutions:

- what does it mean *"knowledge"*?
- what are the demands from *languages* used for knowledge representation?

There is no universally accepted definition of knowledge. Most likely the cause of it lies in very big diversity of the notion. Knowledge (in its intuitive meaning) can be *descriptive* or *procedural*, *explicit* or *tacit*, *qualitative* or *quantitative*, *individual* or *collective*, and so on (even *eastern* or *western*). It is essential to determine the need that the knowledge must fulfill. It can be used for problem solving and decision support, for inference, as an interlingua for cooperating agents or software modules, to define semantics for natural language interpretation, etc.

If the range of knowledge application is limited to knowledge engineering and management, the following definition seems to be appropriate: *knowledge is a special kind of information able to transform the source information to the information in demand*.

In another words – it is a mapping:

KNOWLEDGE: *information* → *information*.

The definition of *information* can be taken from C. Shannon, the father of information theory: *information is everything that decreases uncertainty*.

It is assumed that the meaning of *uncertainty* is well known.

Knowledge as a means of information transformation is well suited to different domains:

- knowledge of problem solving: *problem* → *solution*,
- knowledge of decision support: *problem* + *criteria* → *decision*,
- knowledge of inference: *conditions* → *conclusion*,
- knowledge of search: *demands* → *goal*, etc.

Source and resulting information can be interpreted as input and output data, and knowledge – as a program used for data processing. Such a program should be described by specialized languages of knowledge representation.
Demands for representation language depends on the layer of processing [1, 2].

- The *domain layer* concerns declarative knowledge about the domain of application. Such knowledge describes the objects of discourse in a particular domain, facts that hold about these objects, and relationships among them. This type of knowledge is often represented by concepts, relations, hierarchies, properties, rules, etc. A crucial property of this category of knowledge is that it is represented as much as possible independently from how it will be used.

  The domain layer describes the formal model of a domain.
- The *inference layer* describes the roles of domain expressions and specifies how these expressions are to be used in the inference steps. One can say that the inference layer is a meta-layer of the domain layer.
- The *task layer* enforces control over the inference steps specified at the inference layer. Here it is decided in which order the inferences should be executed. The procedural knowledge is used on this layer and is concerned with sequences, actions, iterations, etc.

In this hierarchy of layers only the domain layer formally describes a particular domain of application and should be adapted to textual form of knowledge sources. That is why only languages of the first layer will be considered below.

Knowledge has many forms that have to be represented by appropriate languages but the most important and frequently used are three groups of demanded dependencies between elements of representation.

- *Logical dependencies* between *statements* with values TRUE or FALSE determine principles for rule based reasoning, the most popular method of inference. The long history of logics and its established position as a tool for formal analysis help to solve numerous problems.
- *Hierarchical dependencies* organize *objects* of a domain into class taxonomy. The ability to represent the relationships between an object and its class or between a class and its superclass has proven to be very useful in many applications. In some cases properties of a class are passed on to its subclasses, simplifying representation.
- *Relational dependencies* between *concepts* represent connections linking together either concrete or abstract elements of the domain. This can help in searching for complex dependencies between given concepts.

There are numerous languages describing some or all of these dependencies. Short presentation of selected, representative examples will help in farther considerations.

2. Languages for knowledge representation

A common problem during the development of ontologies and languages is a range of their applications. It is more costly to create representations that will be reusable across multiple domains than it is to create a language that is suitable for just one application. Languages described below attempt to be more or less universal. The original terminology from the source papers is preserved.

(ML)². A formal language $(ML)^2$ [2] has been developed for the representation of KADS [1] models of expertise. It uses an extended first order predicate calculus. Concepts are represented by constants, i.e., nameable and distinguishable entities. All constants and variables have associated sorts (types, classes), organized in a subsort hierarchy. Predicates and functions are also typed. The relation between constants and sorts is the equivalent of the IS-A relation. A knowledge base is divided into *theories*, consisting of import relations, signature (sorts, constants, models of functions and predicates), variables and axioms (instantiations of predicates, production rules and functions).
The main ideas can be illustrated on the following example:

theory carFailure
  signature
    sorts number (vehicle (bus car (station-car limousine))); *the subsorts tree*
    constants
      myCar, yourCar : car;
      bus-412 : bus;
    functions
      noOfWheels : vehicle → number;
    predicates
      sameBrand : car × car;
      greater : number × number;
  variables a, b, c : number;
  axioms
    same Brand(myCar, yourCar);
    greater(a, b) ∧ greater(b, c) → greater(a, c);
    noOfWheels(yourCar) = 4;
endtheory

Telos. A hybrid language Telos [3] supports three different representation formats: a logical, a graphical (semantic network with relations) and a frame representation. A Telos knowledge base is a finite set of interrelated *propositions* presented as vectors: $\langle oid, x, l, y, tt \rangle$. Their meaning is as follows:

> The object x has a relationships called l to the object y. This relationship has identifier oid and is believed by the system for the time tt.

A standard way to describe objects together with their classes, subclasses and attributes is the frame syntax. It groups the labels of propositions around the label of common source. If $l \in \{*instanceof, *isa, ...\}$ then the triplet $\langle x, l, y \rangle$ is interpreted as $\langle object, relation, class \rangle$. All other propositions describe aggregates $\langle object, attribute, value \rangle$.

Logical operations are performed on such aggregates. There is also a natural interpretation of a set of propositions as a directed graph (semantic network).

CycL. It is a formal language [4] whose syntax derives from first-order predicate calculus and from Lisp. The vocabulary of CycL consists of *terms*, combined into meaningful *expressions (sentences)*. Constants are terms that can denote collections of other concepts such as individual things, classes, predicates and functions. Variables (e.g., ?*X*) stand for constants whose identities are not specified. Sentences have the structure of a Lisp list and consist of objects: predicates and their arguments, logical connectives and quantifiers. Predicates describe taxonomy relations (e.g., *isa*), attribute names and many other relations or properties.
The following is a example of CycL expression:

```
(implies
    (isa ?A Animal)
    (thereExists ?M
        (and (mother ?A ?M)
             (isa ?M FemaleAnimal))))
```

ICONS. The intelligent content management system (ICONS) [5] describes an ontology by *the data model*, that allows to depict the "world", and *the knowledge representation language*, through which one can extend the entities to the schema, infer new knowledge and make some reasoning. The data model contains a set of class names, described by attribute-class pairs and linked to another class name by the *isa* or *partOf* relation, for example:

> *student(name : string, age : integer, enrol : faculty) isa person.*

The language contains appropriate atoms (facts) with the concrete values:

> *student(name : peter, age : 21, enrol : cs).*

Internal class names can be substituted by the full atoms, giving a sequence of dependencies. In production rules with such atoms disjunctions are allowed in the rules' heads (so called disjunction logic programming), for modeling incomplete knowledge. ICONS introduces also reasoning under uncertainty (using Dempster-Shafer theory) and XML as a serial syntax definition language.

F-Logic. In a deductive, object oriented language F-Logic [6, 7] *objects* are organized in *classes* and *methods* represent relationships between objects. Facts are collections of objects with classes and appropriate methods, e.g.,

```
peter:student[father->john:man].
```

Rules use similar forms, often with quantifiers and variables:

```
FORALL X,Y  X[son->Y] <-Y:man[father->X].
```

A query to the object base is formulated as follows:

```
FORALL X,Y   X:women[son->Y[father->john].
```

Properties of objects can be described by predicates, objects denoting numbers are processed with comparison and several arithmetic operators, special path expressions are used to navigate through the large sets of objects.
The presented examples of languages for knowledge representation, and many not presented here (e.g., SILO, CLASSIC, ALLNR, Gellish, GOL, EULE, KIF, ALUNI) show that:

- object-based approach is very popular,
- predicates and first-order logic are basic descriptions of relations and properties,
- the forms of description are diversified,
- not all systems use a taxonomy,
- there are no special approaches simplifying conversion from the unstructured textually-represented knowledge to the formal representation.

The next section tries to collect more important properties of languages and to add textually-oriented elements, in order to construct a useful system of representation.

3. The ontology for textual knowledge

3.1. Compositional representations

The main task of these considerations is to find a way for conversion from unstructured, textual knowledge description to structured, formal knowledge representation. Such a structured form enable processing by computers, reuse in many applications and formal validation.
The conversion should not be to complicated and the structured, final representation should be comprehensible. Therefore a textual form of information should be preserved, if possible, through the whole process of conversion.
Representations are easier for processing and more reusable if they are compositionally constructed. A representation is *compositional* if it describes each individual concept in the domain of discourse, and the representation of complex

concepts is obtained by composing representations of individual concepts. The following example illustrates this, starting from the text:

```
The form PIT is intended for tax-payers
that obtain salaries, pensions
 or scholarships.
```

The noncompositional representation of this might be

THE FORM PIT IS INTENDED FOR TAX-PAYERS THAT OBTAIN SALARIES, PENSIONS OR SCHOLARSHIPS.

This statement can be interpreted as an individual concept or as a logical *proposition* with the truth value (e.g., TRUE or FALSE).

The simple decomposition gives

"THE FORM PIT IS INTENDED FOR *tax-payer*"
if "*tax-payer* OBTAINS SALARY, PENSION OR SCHOLARSHIP".

Now the two *propositions with variables* are connected by the logical operator **if**, giving *decision rule* that can be used for *backward reasoning.* This form is equivalent to the logical rule

$$Q(x) \Leftarrow P(x)\,.$$

Another possible compositional representation can be described, using the model

$$Q(x) \Leftarrow P_1(x) \vee P_2(x) \vee P_3(x)\,,$$

by the following two statements:

THE FORM PIT is intended for TAX-PAYERS GROUP,
person is a member of TAX-PAYERS GROUP **if**
[(*person* obtains SALARY) **or** (*person* obtains PENSION) **or** (*person* obtains SCHOLARSHIPS)].

The elements of these statements have the special meanings:

THE FORM PIT, SALARY, PENSION, SCHOLARSHIPS – *object names* of the domain,
person – *variable* representing an object,
TAX-PAYERS GROUP – *class name* indicating a class of objects,
is intended for, obtains – *roles* played by objects,
is member of – *relation* between object and class,
if, or – logical *operators*.

This representation of knowledge, contained in the source text, can easily be used for reasoning, formally performed by any agent. Having information (*the fact*):

```
Mr Brown obtains pension
```

and substituting MR BROWN for *person,* one can get the conclusion:

MR BROWN IS A MEMBER OF TAX-PAYERS GROUP

and information that the form PIT is for him.

The second example shows some other elements of structural representation.

```
Wanted is employee from Warsaw, 20 to 30
 years old, with MCS degree.
```

The compositional description of this is as follows:

"WANTED EMPLOYEE" [address: WARSAW, age: 20-30, degree: MCS].

This structure contains *object name* and its *properties* in the form of *attribute-name: attribute-value* pairs. It can be considered as a special case (or *instantiation*) of a more general description:

EMPLOYEE [address: TOWN, age: NUMERICAL INTERVAL, degree: DEGREES].

Now the properties of a *class* are presented as *attribute-name: attribute-type* pairs, showing some restrictions for attributes. The object "WANTED EMPLOYEE" *inherits* attribute names and types from the class EMPLOYEE.
The inheritance is sometimes continued, involving more general classes. For example the class EMPLOYEE may be considered as a subclass of the superclass PERSONS, inheriting wanted attributes: general (e.g., nationality: COUNTRY or concrete (e.g., number of legs: 2).
Presented examples show some possible structures, applicable to a knowledge in textual form. To be useful these structures should be described more formally.

3.2. Principles of structuring

Basic elements. Knowledge description language, as all other languages, has many levels of a syntax. The lower one (neglecting numerals and letters) usually contains numbers and strings, but here we will introduce more specialized elements:

- *numbers*;
- *symbols* – like $x, P_1(x), \leq$, USA – used in expressions, as abbreviation or shortening;
- *names* – e.g., employee, Mr Brown – short descriptions, more informative than symbols;
- *texts* – as "MR BROWN OBTAINS PENSION" – easy to understand descriptions of concepts, comments, source knowledge, etc., of special importance in this paper.

Symbol, name and text can represent *constant, variable, relation* and *function*.

Statements. Textual variable with a value from the set of truth values will be called *proposition*. Textual function with such values is known as *propositional function*. Sometimes propositions describe relations, but it is easier to represent relations between constants, variables and functions using conventional symbols ($<, \leq, \subset, \ldots$) or names (*lower, cheaper*).

Some propositions have internal structure enabling shorter and more elastic representation. For example the same meaning as the proposition

MR BROWN HAS VERY HIGH INCOME

has the triplet ⟨MR BROWN, *income*, "very high"⟩

with a general structure

⟨ *object-name*[1], *attribute-name, attribute-value* ⟩.

Such an *aggregate* has truth value (as a proposition), its components can exist as variables, and the form can be modified.

A truth value is a common property of proposition, relation and aggregate, which together will be called *statements*. They constitute basic elements of *logical expressions*: statements connected by logical operators.

Frames. *An object name* (or shortly *object-n*) denotes an element of the domain, physical (as BUILDING) or abstract (as PROGRESS). Properties of an object are described individually in *aggregates* or collectively in *frames*. A simple *object-frame* contains information about an individual concept and has a structure:

⟨*object-n* [*attr-name*$_1$: *attr-value*$_1$,..., *attr-name*$_k$: *attr-value*$_k$]⟩

and all components defined as constant names and values. These names and values belong to certain families, that should be defined.

The family of objects with common properties is called *class* and is described by a *class-frame* (or *prototype frame*) with a structure as follows:

⟨*class-n* [*attr-name*$_1$: *attr-type*$_1$,..., *attr-name*$_k$: *attr-type*$_k$]⟩.

The *type* of attribute denotes a category of its values as:

– the set of concrete values, e.g., 5-9, Mon.-Fri., {white, red}, 2004;

– the class of values, e.g., CITIES, "ALL EVEN NUMBERS", strings.

An object-frame is *instance-of* a class-frame with the same set of attribute names, what is indicated by the structure:

⟨*object* [...] ⟩ *instance-of class*.

Usually each class is a *subclass* of one or more classes higher up in the hierarchy, called its *superclass(es)*. The relations between hierarchically organized classes, showing specialization and generalization, can be presented in the structure:

⟨ *class-a* [...] ⟩ ρ *class-b*

with $\rho \in \{$*is-a, part-of, member-of, ...*$\}$.

[1]In all cases described below the word *text* can be used instead of *name*.

Relation *is-a* causes *inheritance* of all properties from superclass to subclass. In the case of *part-of* relation only some special attributes are inherited (e.g., possession).

Properties of class-frames sometimes include also additional information concerning attributes (units of measure, procedures used for value calculation, etc.).

Dependencies. In many cases objects and classes are related, but not organized in taxonomy. Such "horizontal" relations are known as *roles* and described in the following form, called *dependence*:

⟨*object-a - role - object-b* ⟩.

For example, if A. Smith is a manager of B. White then the appropriate dependence will be:

⟨B. WHITE – manager – A. SMITH ⟩.

Dependencies describe relations, therefore have also truth values and should be included in the set of statements.

Dependencies can be joined in *path expressions* in the form:

⟨*object-a - role-k - object-b - role-l - ... - object-z* ⟩,

as in the example:

⟨B. WHITE – manger – A. SMITH – english-teacher – C. BLACK ⟩.

The set of path expressions with some common objects describes a graph of dependencies, equivalent (if names of objects and roles are properly chosen) to *semantic net* or *conceptual graph*, typical tools of knowledge representation.

If dependencies describing the domain are stored in the knowledge base, path expression with a variable (in the place of an object or a role) can be used for navigation and searching for appropriate value.

Objects and roles in dependencies are presented as names or texts, directly describing some parts of a source document.

Rules. The *logical expression* contains statements connected by logical operators **and, or, not**. The special case of it – *and-expression* – contains only **and** operator. *Production rules* are described by one of the following forms:

if *logical-expression* **then** *and-expression*
if *logical-expression* **then** *and-expression-A* **else** *and-expression-B*
and-expression **if** *logical-expression*.

Different types of a statement (proposition, relation, aggregate, dependence) help in the rule adaptation to various applications.

4. Conclusion

Since statements obtained from the source, textual knowledge have also textual form, then frames, dependencies

and rules, constructed from statements, contain structured texts as well, according to the goal of these considerations.

References

[1] B. Wielinga *et al*., "KADS: a modelling approach to knowledge engineering", ESPRIT Project P5248. Amsterdam: University of Amsterdam, 1992.

[2] F. van Harmelen and J. Balder, "$(ML)^2$: a formal language for KADS models of expertise", *Knowl. Acquis. J.*, vol. 4, no. 1, pp. 127–161, 1992.

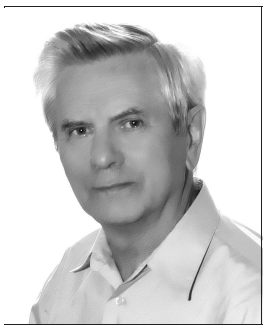
[3] "Frame and network representation", www-i5.informatik.rwth-aachen.deĆBdoc

[4] "Ontological engineer's handbook", www.cyc.com/handbook

[5] "The ICONS knowledge representation features", www.icons.rodan.pl

[6] M. Kifer *et al*., "Logical foundations of object-oriented and frame-based languages", *J. Found. ACM*, no. 42, pp. 741–843, 1995.

[7] *F-Logic Tutorial.* Karlsruhe: Ontoprise GmbH, 2003.



Wiesław Traczyk is a Professor of the National Institute of Telecommunications and also of the Warsaw University of Technology, Institute of Control and Computation Engineering. His research interests include expert systems, approximate reasoning, failures in computer networks and data mining.

e-mail: W.Traczyk@itl.waw.pl
National Institute of Telecommunications
Szachowa st 1
04-894 Warsaw, Poland

e-mail: traczyk@ia.pw.edu.pl
Institute of Control and Computation Engineering
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland