TITLE**:** ***Knowledge Representation for Intelligent Systems***

AUTHOR INFORMATION:

Author: ***Emil Vassev***
Address:
LRG-015, Lero - the Irish Software Engineering Centre,
International Science Centre,
University of Limerick, Limerick, Ireland
Phone: *+353-6-123-3707*
Email: *emil.vassev@lero.ie*
Web: www.vassev.com

Knowledge Representation for Intelligent Systems

Emil Vassev
Lero - The Irish Software Engineering Research Centre,
University of Limerick, Limerick, Ireland
emil.vassev@lero.ie

Abstract

Knowledge is a source of intelligence and both knowledge representation and knowledge management are crucial for intelligent systems. Well employed knowledge helps such systems become aware of situations, recognize states and eventually respond to changes. An intelligent system shall incorporate knowledge that helps the system reason about important external and internal factors, and eventually learn from experience and infer missing knowledge. This entry discusses the notion of knowledge and surveys knowledge representation approaches together with concepts and techniques for system intelligence based on knowledge awareness. Moreover, a brief discussion on the author's view of knowledge representation together with a case study is presented.

Keywords: *knowledge representation; awareness; intelligent systems.*

1. Introduction

In general, an intelligent system is intended to possess self-awareness capabilities based on well-structured knowledge and algorithms operating over the same. Knowledge representation and management is one of the important aspects of developing intelligent systems. Knowledge helps systems achieve awareness and autonomic behaviour, where the more knowledgeable systems are, the closer we get to real intelligent systems. Here, the term "knowledge" is widely used in practice assuming rather vague distinctions among *data*, *information*, and *intelligence*. Along with such a context, any discussion on knowledge should refer to those categories. By its nature as a source of intelligence, knowledge allows system to recognize its states and helps to decide how to respond to situations. Pejman Makhfi [1] concludes that the concept of intelligence is built upon four fundamental principles: *data*, *information*, *knowledge*, and wisdom. In this quartet, the basic compound for intelligence is data. In general, data takes the form of measures and representations of the internal and external worlds of a system, e.g., raw facts and numbers. Information is derived from data by assigning meaning to it relevant to the domain of interest, e.g., data in a specific context. Knowledge is a specific *interpretation* of information and wisdom is based on *awareness*, *judgment rules* and *principles* of constructing new knowledge from existing one.

The rest of this entry is organized as follows: Section 2 presents kinds of knowledge that should be considered when developing intelligent systems. Basic knowledge-representation approaches are

presented in Section 3. Section 4 discuses important factors and challenges to knowledge representation, such as completeness, consistency, models and reasoning. Section 5 presents a brief case study of knowledge representation. Finally, Section 6 concludes the chapter with a brief summary.

2. Kinds of Knowledge

There are many kinds of knowledge that need to be considered for the development of intelligent systems. Conceptually, knowledge can be regarded as a large complex aggregation [2] composed of constituent parts representing knowledge of different kind. Each kind of knowledge may be used to derive knowledge models of specific domains of interest. For example, in [2] the following kinds of knowledge are considered:

- *domain knowledge* – refers to the application domain facts, theories, and heuristics;
- *control knowledge* – describes problem-solving strategies, functional models, etc.;
- *explanatory knowledge* – defines rules and explanations of the system's reasoning process, as well as the way they are generated.
- *system knowledge* – describes data contents and structure, pointers to the implementation of useful algorithms needed to process both data and knowledge, etc. System knowledge also may define user models and strategies for communication with users.

Moreover being considered as essential system and environment information, knowledge may be classified as 1) *internal knowledge* - knowledge about the system itself; and 2) *external knowledge* - knowledge about the system environment. Another knowledge classification could consider *a priori knowledge* (knowledge initially given to a system) and *experience knowledge* (knowledge gained from analysis of tasks performed during the lifetime of a system). Therefore, it depends on the problem domain what kinds of knowledge may be considered and what knowledge models may be derived from those kinds. For example, we may consider knowledge specific to: 1) the internal system component structure and behaviour; 2) the system-level structure and behaviour; 3) the environment structure and behaviour; and 4) different situations where a system component or the system itself might end up in. In addition, knowledge of components' and system's capabilities of communication and integration with other systems may be considered as well.

3. Knowledge Representation

Different knowledge representation techniques might be used to represent different kinds of knowledge and it is our responsibility to pick up ones that suit our needs the most. In general, to build a knowledge model we need specific knowledge elements. The latter may be primitives such as *frames, rules, logical expressions*, etc. Knowledge primitives might be combined together to represent

more complex knowledge elements. A knowledge model may classify knowledge elements by type and group those of the same type into collections.

Different approaches to knowledge modelling and knowledge representation have been developed for intelligent systems. Note that it is important to structure the knowledge in such a way that in can be effectively processed by an intelligent system and perceived and updated by humans. The following subsections present some of the most popular approaches to knowledge representation [3].

3.1. Rules

Rules can be easily understood by humans. By its nature, rules structure knowledge in the form of *attribute-value pairs*. In general, rules may take the following form [3]:

if attribute A1 has value V1
and attribute A2 has value V2
then attribute A3 has value V3

The attribute part of a rule can consist of a series of clauses where the AND, and to a lesser degree OR and NOT, logical connectives are possible. A rule that contains two clauses in its attribute part connected by logical OR can be re-written as two rules. For example, the following rule:

IF it is lunchtime OR I am hungry THEN I shall go to the restaurant.

might be presented as:

IF it is lunchtime THEN I shall go to the restaurant.
IF temperature is high THEN ice-cream sales are high.

Sometimes, rules might have the form of "IF premise THEN action", where "premise" is Boolean and "action" is a series of statements. Note that although similar, there are some important differences between the *rules used for knowledge representation* and the conditional statements found in conventional programming languages:

- Rules are relatively independent of one another.
- Rules can be based on heuristics or experiential reasoning.
- Rules can accept uncertainty in the reasoning process.
- Conditional statements combine the knowledge and reasoning in one structure.
- Rules simply represent the knowledge, i.e., the inference mechanism or reasoning is separate.

Shortliffe successfully applied the rule-based approach to the development of systems applying human knowledge and function at the level of a human expert [4]. In this approach, attributes may

represent internal data and both system input and output (e.g., a response from the user). In such a knowledge model, it is relatively easy to construct an engine that uses the set of rules in an automated reasoning system. Rules may be dynamic, i.e., they can be archived and updated as necessary.

The so-called *exception systems* use a similar approach to knowledge representation. Here, the rules may take the following form [3]:

attribute A1 has value V1
unless attribute A2 has value V2
and attribute A3 has value V3

One of the main advantages of the rule-based knowledge representation is the extreme simplicity of the model, which helps us easily understand the knowledge content. The explanation for the reasoning is easily shown, i.e., a list of rules that fire under specific conditions. However, a rule-based knowledge representation may grow very large incorporating thousands of rules, which require extra efforts and tools to maintain their consistency.

3.2. Frames

Frames is another approach to knowledge representation understandable by humans. Frame-based knowledge models represent simple concepts via a collection of information and associated actions. Often, the notion of frame is related to data structure containing typical knowledge about a *concept* or *physical entity* – an object, a person, ect. An example of a simple representation of a person with the frame-based approach is the following [3]:

Frame: Ellery Stone
Specialisation of: Frame Person
Date of Birth: 30:04:62
Sex: Male
Nationality: British
Home Town: St. Helens
Occupation: Tailor
Health: (Consult Medical system)

Frames represent knowledge about real-world entities by combining information, calls to information derivation functions and output assignments. A frame contains both descriptions of attributes and procedural details. As shown in the simple frame above, some of the *slots* have associated values and one slot refers to another system that must be used to find a value. A frame can encompass both semantic and procedural knowledge. Usually, expert (or knowledge-base) systems using frames as their fundamental knowledge representation scheme are referred to as *frame-based systems*. In a frame we recognize two key elements: 1) *slots* - sets of attributes for specific entity that

is described; and 2) *facets* - extended knowledge about an attribute in a frame. By relaying on these two key elements, a frame may express knowledge in the following ways:

- *value* – a simple attribute value (can be symbolic, numeric, Boolean, etc);
- *default* – the value taken if the attribute is not otherwise described;
- *range* – determines what type of information can appear in the attribute;
- *demons* – usually have an IF-THEN structure and allow *procedural knowledge* to be combined with the *declarative knowledge* stored in the frame, e.g.,:
  - *if-added* – determines what action is to be undertaken when a value is added to an attribute;
  - *if-needed* – determines what procedure runs to obtain a value if such is needed.

One of the main advantages of frames is the ability to include *demons* to compute slot values. A demon can run a function that computes the value of a slot on demand.

3.3. Semantic Networks

The third of the basic approaches to knowledge representation is termed as *semantic networks* [5]. Similar to the previous two approaches, semantic networks provide a knowledge representation style that humans can easily cope with. The idea behind a semantic network is that knowledge is often best understood as a set of concepts that are related to one another. Basically, a semantic network is a directed graph consisting of *nodes* (or vertices) connected with *edges* (or arcs). Nodes represent *concepts* and edges represent *semantic relations* between those concepts. There is no standard set of relations between concepts used in semantic networks, but the following relations are very common:

- *instance*: X is an *instance* of Y if X is a specific example of the general concept Y.
  Example: Object A is an *instance* of Class B
- *isa*: X *isa* Y if X is a subset of the more general concept Y (see Figure 1).
  Example: sparrow *isa* bird
- *haspart*: X *haspart* Y if the concept Y is a part of the concept X.
  Example: sparrow *haspart* tail

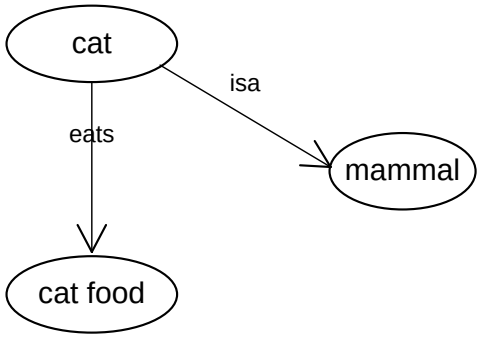

Figure 1. Semantic Network Example

Inheritance is a key notion in semantic networks and can be represented naturally by following *isa* relations. In general, if concept *X* has property *P*, then all concepts that are a subset of *X* should also have property *P*. In practice, inherited properties are usually treated as default values. If a node has a direct link that contradicts an inherited property, then the default is considered *overridden*.

Essentially, a computer-based semantic network uses metadata (data describing data) in order to represent the meaning of different information. Here, metadata helps an intelligent system understand the meaning of information. Note that systems able to recognize the meaning of information (e.g., stored in a data warehouse) become immeasurably more intelligent. Extensible Markup Language (XML) and Resource Description Framework (RDF) are content-management approaches supporting semantic networks. XML and RDF are able to sort through vast amounts of data automatically, recognising relationships between information and presenting high-quality, relevant data to the user on demand. XML lets us create our own tags - hidden labels such as <*cat*> or <*mammal*> that annotate semantic network nodes and edges. The meaning of the nodes is expressed by RDF, which encodes it in sets of triples: *resource*, *property* and *value* (similar to the *subject*, *verb* and *object* of an elementary sentence expressed in a natural language). RDF gives particular entities (expressed with nodes) properties such as "is a member of" with certain values, e.g., another entity (person). For example, the sentence "*John is a member of the golf club.*" can be expressed with RDF and XML as:

```
<rdf:Description about= the golf club>
    <Member> John </Member>
</rdf:Description>
```

The so-called Simple Knowledge Organization System (SKOS) is a knowledge representation technique that provides RDF with a model for expressing the basic structure and content of concept schemes [6]. The latter are defined as a set of concepts with optional semantic relationships between the concepts. SKOS is implemented as an XML namespace providing properties helping to represent a concept scheme. For example, SKOS uses "skos:broader" and "skos:narrower" to represent semantic relations and "skos:broaderGeneric" and "skos:narrowGeneric" to have class subsumption semantics.

A special form of a semantic network is the *semantic web*. The semantic web is a collaborative effort led by World Wide Web Consortium (W3C) that aims to transform the way we find information stored on the Internet. Most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully, i.e., a computer browsing over the Internet has no reliable way to process the semantics of the webpages' content. Rather than search for information containing specific keywords, the semantic web shall help us search for the semantic meaning of the content. The nature of the semantic network knowledge representation techniques is such that

additional data can immediately be added to existing knowledge bases, helping search engines retrieve much more relevant information and giving instant value to useful user-created content.

The so-called Friend of a Friend (FOAF) [7] representation technique is a sort of *dictionary* for RDF providing support to defining connections (relations) in the semantic web. The idea behind is to allow searching over the web for documents based on their properties and their inter-relationships. Similar to SKOS, FOAF is defined as an XML namespace and can be used in the web pages as part of RDF to define valuable properties of those web pages. For example, the "foaf:maker" property may be used to identify the creator of an RDF page:

```
<foaf:maker>
    <foaf:Person>
        <foaf:mbox rdf:resource="mailto:myname@mymailserver.com" />
    </foaf:Person>
</foaf:maker>
```

Semantic databases are relevant knowledge representation structures relying on the semantic network approach to organize data so that the data elements are related to each other based on their meaning [8]. Similar to rational databases, semantic databases also provide for separation of the logical data from the physical implementation, but they target more powerful abstractions for the specification of database schemas than the rational databases do. Semantic databases don't have fixed schemas, which leads to greater flexibility in the database design. Data carries its meaning in a semantic network and can be interpreted directly by the users. The data representation model naturally supports a top-down modular view of the schema, thus simplifying both schema design and usage. As a result, the semantic database query engines know more about the meaning of the stored data and about the meaningful data connections. Ideally, this knowledge can be applied to perform faster database operations, e.g., more efficient and straightforward drill-down.

The semantic databases use SQL-like query languages, which may use FOAF properties as parameters. The following is a simple SELECT query written in such a language called SPARQL (a query language for RDF) [9]:

```
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```

Ideally, querying should be deductive (inferential), however the current implementations can only cope with very inexpressive knowledge bases, barely providing little more than just taxonomies.

3.4. Concept Diagrams

Another approach to knowledge representation is the so-called *concept diagrams* (also called *concept maps*). By their nature, concept diagrams are very similar to semantic networks, i.e., they also consist of nodes and arcs (links), where the nodes and arcs have similar meaning and functions. However, in contrary to semantic networks, concept maps allow the links between the nodes to be labelled in very different ways. Moreover, concept diagrams are considered more powerful, because they can describe fairly complex concepts, e.g., hierarchy of concepts where a node can be a concept diagram itself (see Figure 2).

A simpler view of cognitive cartography has also been proposed under the term of *mind maps* [10] where only concepts and their proximity are represented, without any particular meaning imposed on the relationships. Concept maps are important when we want an intelligent system adopt a constructivist view of learning. The theory behind it is that the system develops a sort of "mental schema" or mind maps, which serve to inform future thinking or action.

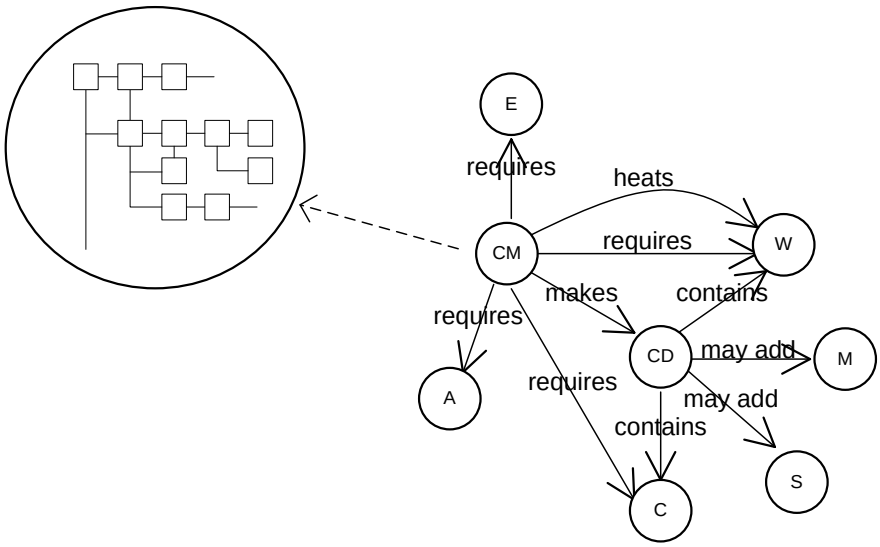

Figure 2. Concept Diagram Example

The Institute for Human and Machine Cognition (IHMC), Florida, USA developed a special form of concept diagram to represent a domain of knowledge [11]. The knowledge embedded in a knowledge model is structured in the form of *concept maps* [12]. Concept maps are graphs that are comprised of concepts on the nodes and the relationships among the concepts on the arcs. Concept maps are used to form knowledge models by placing them in a hierarchical organization and appending special elaborating media onto the nodes within each map. The entire knowledge model is linked together through a general, subsuming a top-level map. The result is a model of expert knowledge that contains numerous of domain concepts, principles, and relations.

3.5. Hierarchy of Concepts

The so-called Autonomic System Specification Language (ASSL) [13] implies another hierarchical approach to knowledge representation for intelligent systems. ASSL exposes a *hierarchical knowledge model* (also called *multi-tier specification model*) defined through formalization tiers forming a special hierarchy of concepts. Those tiers provide a judicious selection and configuration of infrastructure elements and mechanisms needed by an autonomic system (AS) [14]. The latter is considered as being composed of special autonomic elements (AEs) interacting over interaction protocols, whose specification is distributed among the ASSL tiers. Note that each tier is intended to describe different aspects of the AS in question, such as *service-level objectives*, *policies*, *interaction protocols*, *events*, *actions*, etc. This helps us specify an AS at different levels of abstraction imposed by the ASSL tiers.

```
I. Autonomic System (AS)
  * AS Service-level Objectives
  * AS Self-managing Policies
  * AS Architecture
  * AS Actions
  * AS Events
  * AS Metrics
II. AS Interaction Protocol (ASIP)
  * AS Messages
  * AS Communication Channels
  * AS Communication Functions
III. Autonomic Element (AE)
  * AE Service-level Objectives
  * AE Self-managing Policies
  * AE Friends
  * AE Interaction Protocol (AEIP)
    - AE Messages
    - AE Communication Channels
    - AE Communication Functions
    - AE Managed Elements
  * AE Recovery Protocol
  * AE Behavior Models
  * AE Outcomes
  * AE Actions
  * AE Events
  * AE Metrics
```

Figure 3. ASSL Multi-tier Knowledge Model

Figure 3 presents the multi-tier knowledge model of ASSL. As shown, the latter decomposes an AS in two directions 1) into levels of functional abstraction; and 2) into functionally related tiers (sub-tiers). With the first decomposition, an AS is presented from three different perspectives, these depicted as three main tiers (main concepts):

- AS Tier forms a general and global AS perspective exposing the architecture topology, general system behaviour rules, and global *actions*, *events*, and *metrics* applied to these rules.

- ASIP Tier (AS interaction protocol) forms a communication perspective exposing a means of communication for the AS under consideration.
- AE Tier forms a unit-level perspective, where an interacting sets of the AS's individual components is specified. These components are specified as AEs with their own behaviour, which must be synchronized with the behaviour rules from the global AS perspective.

3.6. Ontologies

Ontologies inherit the basic concepts provided by *rules* (see Section 3.1), *frames* (see Section 3.2), *semantic networks* (see Section 3.3) and *concept diagrams* (see Section 3.4) and provide a form of explicit representation of domain *concepts*, *objects* and the *relationships* between those concepts/objects to form the basic structure around which knowledge can be built [15]. The main idea is to establish *standard models*, *taxonomies*, *vocabularies* and *domain terminology*, and use those to develop appropriate knowledge and reasoning models. Such models may be used as reusable components for assembling knowledge systems, e.g., multi-agent systems. Any ontology is a *formal* and *declarative representation* of a knowledge model of some topic or subject area. It provides concepts to be used for expressing knowledge in that subject area. This knowledge encompasses: *types of entities*, *attributes* and *properties*, *relations* and *functions*, as well as various *constraints*.

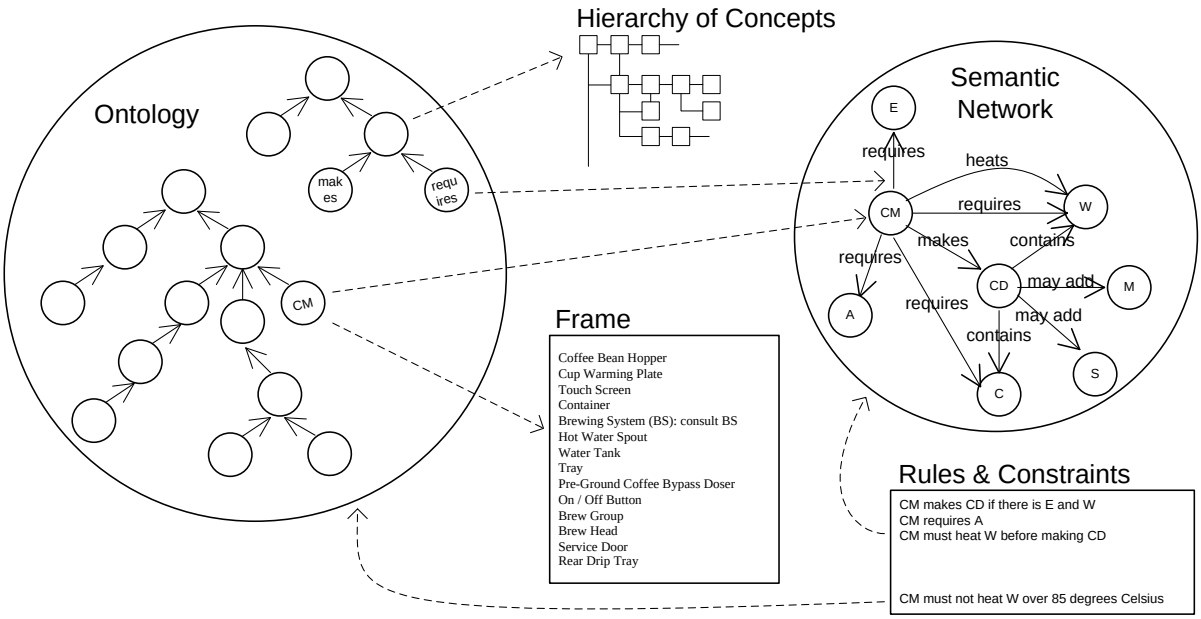

Figure 4. Ontology versus Rules, Frames, Semantic Networks and Hierarchy of Concepts

Figure 4 depicts the relationships between the Ontology approach to knowledge representation and the other approaches described in this section. As shown, an *ontology* is composed of *hierarchies of concepts* (e.g., "objects" concept tree and "relations" concept tree). Each concept itself has properties, which can be regarded as a *frame*. The relationships between the concepts form *semantic networks* and

the *rules* and *constraints* impose restrictions over the relationships or define true statements in the ontology (*facts*). For example, a concept represented in that ontology is "Coffee Machine" (CM). CM has properties like: height, weight, coffee bean hopper, touch screen, container, etc. The ontology relations and concepts form a semantic network expressing the relationships between the CM and the rest of the concepts in the ontology. This semantic network has the following properties: *CM requires E (electricity), CM requires A (action), CM requires C (coffee), CM requires W (water), CM makes CD (coffee drink),* etc. In addition, some rules expressed with the ontology concepts add new knowledge about the coffee machine:

- *CM makes CD if there is E and W.*
- *CM must not heat W over 85 degrees Celsius.*

Ontologies are very powerful knowledge representation mechanisms providing inherent inferential techniques that help an intelligent system reason to:

- answers queries over ontology classes and instances (individuals):
  - finds more general/specific classes;
  - retrieves individuals matching a giving query.
- integrate and align multiple ontologies.

In general, to build ontology, a special ontology language is required. For example, the Web Ontology Language (OWL) [16] is such a formal language that evolved out of Description Logic (see Section 3.7) and is the result of research efforts aiming at providing a knowledge representation language for the semantic web. By using OWL, we build an ontology as an explicit and formal specification of a conceptualization that eventually can be compared with a TBox ("T" states for "template" or "terminology") in Description Logic [17], i.e., it provides a vocabulary of concept definitions and defines relationships among these concepts. These concepts in turn are used to represent knowledge about specific objects in the world.

Another ontology language is CycL [18], which is considered as an extension of First Order Logic [19]. CycL has been used to build the Cyc ontology [18], which is structured into different layers and consists of *terms* constituting the ontology's vocabulary and *assertions* that represent relationships between terms. These assertions are *simple ground assertions* and *rules*.

3.7. Logics

To give a knowledge representation approach (e.g., Semantic Networks, Rules, etc.) a precise semantics, the knowledge representation is often formalized using a logic [19]. Without a precise formalisation a knowledge representation is vague and ambiguous and thus, not appropriate for computational purposes. Moreover, logic is relevant to *reasoning*, which is about inferring new knowledge from the existing one, which in logic is relevant to logical entailment and logical deduction

[19]. The most prominent logical formalism used for knowledge representation is the *First-Order Predicate Calculus* or also called *First-Order Logic* (FOL). FOL helps us describe a knowledge domain as consisting of *objects* and construct *logical formulas* around those objects. Such formulas are formed by *predicates*, *functions*, *variables*, and *logical connectives* [19]. Similar to semantic networks, statements in natural language can be expressed with logical formulas describing facts about objects with an appropriate choice of *predicate* and *function symbols*. The following example illustrates the use of FOL for knowledge representation by axiomatizing the Semantic Network example presented in Figure 1:

$\forall x : (\text{Cat}(x) \rightarrow \text{Mammal}(x))$
$\forall x,y : (\text{eats}(x,y) \rightarrow \text{Cat}(x) \wedge \text{CatFood}(y))$

The logic formalism imposes a few reasoning techniques that can be used to infer new knowledge from existing one:

- *induction* - inducts new general knowledge from specific examples;
  Example: Every dog I know has ears. → Dogs have ears.
- *deduction* – deducts new specific knowledge from more general one;
  Example: Dogs bark. Bobby is a dog. → Bobby barks.
- *abduction* – concludes new knowledge based on shared attributes.
  Example: Sherlock Holmes: The murderer was left-handed.
  Smith is left-handed. → Smith is the murderer.

Extensions of FOL such as *Second-Order Logic* and Temporal Logics strive to improve the logical formalism by extending the expressiveness of FOL. The problem with FOL is that we may quantify over individuals, but not over properties and time. With FOL we can find the individuals of a property, but we cannot find the properties of an individual. For example, with Second-Order Logic we can axiomatize the sentence "*component A and component B have at least one property in common, e.g., share at least one interface*", which cannot be done with FOL. In addition, Temporal Logics help us model knowledge either as *linear time* or *branching time* temporal models, both introducing the notion of time in the knowledge system in question. Moreover, Temporal Logics might be used to describe and formalize complex *reasoning patterns* prescribing inference steps operating over temporal knowledge models [20].

Another prominent formalism is the *Description Logic*. Actually the term "Description Logic" stands for a family of knowledge representation languages that evolved from *semantic networks* and *logic* and differ in their expressiveness (e.g., OWL - see Section 3.6). With Description Logic we represent the knowledge of an application domain (the world) by first defining the relevant concepts of the domain (its terminology) and then using these concepts to specify properties of objects and

individuals occurring in the domain (the world description) [17]. A knowledge model expressed in a Description Logic consists of two main categories of knowledge: TBox and ABox. TBox stands for "terminology box" and contains terms and relationships that are important concepts in the domain under consideration. The vocabulary defined in TBox consists of concepts that denote sets of individuals, e.g., objects and roles denoting binary predicates between individuals. Thus, TBox provides a vocabulary that can be used to describe objects and their relationships in the target domain and to specify general knowledge that is further applied to specific situations. This is done in ABox by specifying *special assertions* about concrete concepts or individuals. For example, it could be asserted that "the black cat" is an instance of "Cat" and she is currently "eating". Hence, ABox contains the representation of objects in the world and that representation is constructed with vocabulary that is initially defined in TBox.

An important aspect of Description Logic is the kind of inference mechanisms it supports. Main inference tasks that can be performed over a Description Logic knowledge base are *subsumption*, *classification* and *consistency*. The main ideas behind Description Logic-based knowledge representation are [17]:

- The basic syntactic building blocks are *atomic concepts* (unary predicates), atomic roles (binary predicates) and individuals (constants).
- The expressive power of the language is restricted in the sense that it uses a rather small set of constructors for building complex concepts and roles.
- Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures. In particular, subsumption relationships between concepts and instance relationships between individuals and concepts play an important role.

Considering syntax and semantics of Description Logic, the latter should be considered as a fragment of FOL. Atomic concepts and atomic roles can be seen as unary and binary predicates and any Description Logic formula can be translated into an equivalent FOL formula. Advantages of Description Logic over FOL are that its syntax is more compact and that it has better computational properties at the expense of being less expressive than FOL.

4. Discussion on Knowledge Representation

4.1. Encoded Knowledge versus Represented Knowledge

When working on knowledge representation, it is necessary to consider the fact that most of the time the system under development is not 100% knowledge-centered. This means that software engineers and developers may design and encode a big part of the "*a priory*" knowledge (knowledge given to the system before the latter actually runs) in the implemented program classes and routines. In such a case, the knowledge-represented pieces of knowledge (e.g., concepts, relations, rules, etc.) may complement

the knowledge codified into those program classes and routines. For example, rules could be based on classes and methods and a substantial concern about the rules organization is how to relate the knowledge expressed with rules to implemented methods and functions. A possible solution is to map concepts and objects to program classes and objects respectively and design rules working on the input (parameters, pre-conditions) and output (results, post-conditions) of implemented methods.

4.2. Knowledge Completeness and Consistency

It should be noted also that an essential assumption when building knowledge models is that such cannot provide a complete picture of the domain of interest. The fundamental reasons are that domain objects often present real things that cannot be described by a finite set of symbolic structures. Moreover, such objects do not exist in isolation, but are included in unlimited sets of encompassing contexts. Therefore, incompleteness shall be considered when developing knowledge models and also the fact that an intelligent system must rely on reasoning to infer missing knowledge.

Another aspect of the knowledge completeness is related to the way a system assumes its operational world. In this regard, we may have:

- Closed World Assumption (CWA) – assumes a complete and closed model of the world;
- Open World Assumption (OWA) – assumes an incomplete and open model of the world.

Whereas the CWA strategy assumes that unless an atomic sentence is known to be true, it can be assumed to be false, the OWA strategy assumes that any information not explicitly specified (or such that cannot be derived from the known data) is considered unknown. Note that FOL imposes CWA semantics and Description Logics impose OWA semantics. The following example shows the difference between these two assumptions:

Given: Emil drives Mazda.
Question: Does Emil drive a red Mazda?
Answer: (CWA) No.
(OWA) Unknown.
(The Emil's Mazda could be red.)

Although more restrictive, CWA provides for avoiding inconsistency in knowledge – if consistent, knowledge cannot become inconsistent, because CWA does not allow adding new facts, which may lead to inconsistency. Knowledge consistency is important for efficient reasoning. Some cases of inconsistency in knowledge are presented below:

- *conflicts* between rules, between facts, between relations and between constraints, e.g., two rules have the same conditions but conclude different results;
- *redundancy* in rules, facts and constraints, e.g., two rules are applicable to identical situations and conclude the same results;

- *rule and constraint subsumption* when two rules (or constraints) conclude the same results, but the first is more restrictive than the second one and whenever the first one succeeds the second one succeeds as well.

Other approaches helping a system preserve knowledge consistency are "no negation" and the use of consistency rules and constraints. For example, rules may imply true *facts* or *special relations* between the concepts and objects or impose a special semantics for such relations. As part of the knowledge representation, rules may provide special constraints ensuring that the knowledge will be correctly processed by the inferential engines. There could be constraints for *knowledge acquisition*, *knowledge retrieval*, *knowledge update* and *knowledge inference*.

4.3. Knowledge Models

Conceptually, intelligent system knowledge should be represented in a way suitable for machine learning and data mining. Therefore, data must be structured (or classified) to become a source of knowledge. In general, to represent knowledge, we may consider special *knowledge models* for the single system component (*component knowledge model*), the system as a whole (*system knowledge model*) and the operational environment (*context knowledge model*). In addition, we also may consider specific situations, which may be expressed as special *situational knowledge patterns*. The latter will help a system recognize situations it has been before and eventually detect unknown ones. The following is a brief rational giving some hints about possible knowledge models and techniques we may consider.

There may be different knowledge models, depending on the complexity of the knowledge to be represented. This complexity is also related to the complexity of the system of which knowledge has to be represented. For instance, single components are much less complex entities than the system hosting those components. Thus, it is more reasonable to model knowledge about components (*component knowledge model*) in much more detail than for the system as a whole. Also, because of the architecture of the system in question, it is reasonable to assume that each system component has accurate knowledge of its own situation, while the situation in which a system is would be less certain. For these reasons, we may consider modelling local knowledge of each system component (behaviour, properties and policies) with deterministic models such as *Finite State Machines* or *Logic Formulae*. These models offer methods to accurately determine the current situation of a system component. On the other hand, knowledge about the system or the environment may be encoded into probabilistic models such as *Bayesian Networks*, *Neural Network* or *Support Vector Machines*. Although less accurate, these models offer practical ways for a system component to determine what is the most likely current situation of the system it belongs to. Moreover, probabilistic models are suitable for evolution and adaptation in case unplanned situations arise. This is of great interest when the system is facing an unknown situation. The learning capabilities of probabilistic models make it

then possible to update knowledge representation with new situations. The information provided by the system component knowledge models will help each system component identify the current situation (expressed in the form of situational knowledge patterns) in which a system is.

4.4. Prolog

A good approach to representing knowledge in a rule-based or ontology-based way is to use the Prolog programming language [21]. Prolog imposes a logic programming model (it uses a mathematical logic for computer programming). In general, a Prolog program can be considered as a *knowledge representation* that captures aspects of the knowledge domain under consideration. Main elements of a Prolog program are *facts*, *rules* and *queries*, which makes it appropriate for writing knowledge specifications of rule-based knowledge models and ontologies. Facts express aspects of a world that are unconditionally true. In contrast to facts, rules represent facts that are conditionally true. As common for rules, a rule in Prolog consist of two parts namely a *condition* and a *conclusion*. Since a Prolog program is a knowledge representation, Prolog also provides a mechanism to query the represented knowledge. This is possible through special *queries* that have the same syntax as facts but may be parametrized and provided as queries to the Prolog interpreter.

4.5. Reasoning

When a knowledge system needs to decide on a course of action and there is no explicit knowledge about this, the system must *reason*. Basically, *reasoning* is how a system uses its knowledge to figure out what it needs to know from what is already known. There are two main categories of reasoning:

- *monotonic reasoning* - new facts can only produce additional beliefs;
- *non-monotonic reasoning* - new facts will sometimes invalidate previous beliefs.

Computations over the represented knowledge are done by the so-called *inferential engine*s that act on knowledge to produce new knowledge. Usually, the inferential engines are FOL-based or DL-based. The FOL-based *inferential engines* use algorithms from automated deduction dedicated to FOL, such as *theorem proving* and *model building*. Theorem proving can help in finding contradictions or checking for new information. Finite model building can be seen as a complementary inference task to theorem proving, and it often makes sense to use both in parallel. Some common FOL-based inference engines are VAMPIRE, SPASS, and the E theorem prover. The problem with the FOL-based inference is that the logical entailment for FOL is *semi-decidable*, which means that if the desired conclusion follows from the premises then *eventually* resolution refutation will find a contradiction. As a result, queries often unavoidably do not terminate.

Inference engines based on Description Logics (e.g., Racer, DLDB, etc.) are extremely powerful when reasoning about *taxonomic knowledge*, since they can discover hidden *subsumption relationships*

amongst classes. However, their expressive power is restricted in order to reduce the computational complexity and to guarantee the *decidability* (on Description Logics are decidable) of their deductive algorithms. Consequently, this restriction prevents taxonomic reasoning from being widely applicable to heterogeneous domains (e.g. integer and rational numbers, strings) in practice.

In general, different knowledge-representation mechanisms require different formalisms. It might be considered that *ontologies*, *frames*, *rules* and *semantic networks* are intended to present distinct pieces of knowledge that are worth being differently represented. Note that an important distinction is between *ontological* and *factual* knowledge. Whereas the first is related to the *general categories* (presented as concepts) and important objects in the domain of interest, the second makes *assertions* about some specific concepts and objects. This distinction is essential for reasoning based on Description Logics (DL). In addition, the rules and constraints are part of the so-called *explicit knowledge* suitable for FOL-based reasoning.

4.6. Awareness

Awareness is a concept playing a crucial role in intelligent systems. Conceptually, awareness is a product of *knowledge representation*, *knowledge processing* and *monitoring*. Autonomic computing [14], one of the modern approaches to knowledge systems, considers two kinds of awareness:

- *self-awareness* – a system has detailed knowledge about its own entities, current states, capacity and capabilities, physical connections and ownership relations with other systems in its environment;
- *context-awareness* – a system knows how to negotiate, communicate and interact with environmental systems and how to anticipate environmental system states, situations and changes.

A key success factor for a knowledge system is to employ its knowledge to become an *aware system*. Such a system must be able to sense and analyze its components and the environment where it operates in. A primary task should be determining the state of each system component and the status of the service-level objectives. Thus, an aware system should be able to notice change and understand the implications of that change. Here, both *self-monitoring* and *monitoring* of the environment appear to be one of the key concepts in awareness. A possible approach will be to develop a monitoring system based on notification events. Moreover, an aware system should be able to apply both *pattern analysis* and *pattern recognition* to determine normal and abnormal states. Here, knowledge might be expressed in the form of operational patterns, e.g., performance, resource usage, etc.

Figure 5 depicts a generic model for system awareness based on structured knowledge. As shown, at the core of the awareness model is structured knowledge. This is the knowledge a system maintains in order to be aware of changes taking place in its internal world and in the environment (context knowledge model). Note that system components must have its internal knowledge and possibly have

knowledge about the system as a whole and/or the environment. Moreover, a system must maintain special situational knowledge eventually expressed as *situational knowledge patterns* describing special situations that must be considered by that system. Such situations are relevant changes in the environment or in the system itself. A situation could be a joint situation where changes in different domains of interest are considered. For example, a change in the internal system's structure and a change in the environment might be considered as a situation.

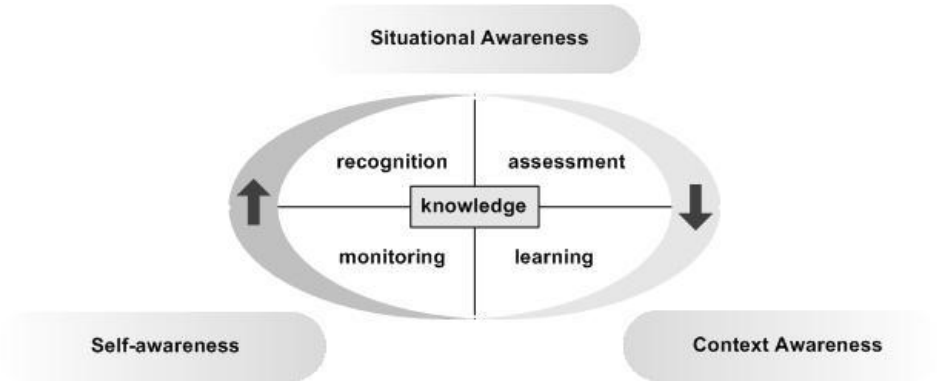

Figure 5. Generic Mechanism for Awareness in Intelligent Service Components

The model for system component awareness comprises a special awareness control loop where we observe four distinct functions:

- *monitoring* - collects, aggregates, filters, manages and reports internal and external details (e.g., metrics and topologies) gathered from the internal entities of a system and from its context;
- *recognition* - uses knowledge models to recognize changes in the system or in the context;
- *assessment* - determines entities (internal or external) of interest, generates hypotheses about situations involving these entities, and recognizes situational patterns;
- *learning* - generates new situational patterns and maintains history of property changes.

Therefore, in order to monitor and collect information consistently and to recognize changes, a system must have a well-structured uniform knowledge model of itself and eventually such of its context. Moreover, it must have situational knowledge patterns that help the system recognize situations. The four functions forming the awareness control loop help a system be aware of internal changes (self-awareness), of external changes (context awareness) and of situations (situational awareness).

5. Case Study – Knowledge Representation for ASCENS

Autonomic Service-Component Ensembles (ASCENS) [22] is a class of multi-agent systems formed as mobile, intelligent and open-ended swarms of special *autonomic service components* capable of local and distributed reasoning. Such *service components* encapsulate rules, constraints and mechanisms for self-adaptation and acquire and process knowledge about themselves, other service components and their environment. ASCENS systems pose distinct challenges to knowledge representation.

5.1. Kinds of Knowledge for ASCENS

There have been determined four basic knowledge domains (kinds of knowledge) for ASCENS systems:

- the individual component structure and behaviour;
- the system structure and behaviour;
- the environment structure and behaviour;
- situations where the system might end up in.

These knowledge domains have been used to derive distinct *knowledge models* for ASCENS forming a *high-level knowledge structure* that is to be maintained by any *service component* (SC) member of a *service-component ensemble* (SCE):

- *SC knowledge model* - knowledge about internal configuration, resource usage, content, behaviour, services, goals, communication ports, actions, events, metrics, etc.;
- *SCE knowledge model* – knowledge about the whole system, e.g., architecture topology, structure, system-level goals and services, behaviour, communication links, public interfaces, etc.;
- *environment knowledge model* – parameters and properties of the operational environment, e.g., external systems, external communication interfaces, integration with other systems, etc.;
- *situational knowledge patterns* - specific situations, involving one or more SCs and eventually the environment.

By representing the knowledge in such models, an individual SC shall be able to query information about both the SC itself and the SCE, by considering the environment's parameters and properties. Moreover, this helps SCs understand and reason about themselves and discover situations through the use of probabilistic methods working over the knowledge modeled as situational knowledge patterns.

5.2. Structure of the Knowledge Representation

The four knowledge domains for ASCENS are represented by four distinct *knowledge corpuses* - SC Knowledge Corpus, SCE Knowledge Corpus, Environment Knowledge Corpus and Situational Knowledge Corpus. Each knowledge corpus is structured into a special *domain-specific ontology* (see Section 3.6), special *contexts* and a *logical framework* (see Section 3.7). The domain-specific ontology gives a *formal* and *declarative representation* of the knowledge domain in terms of explicitly described domain concepts, individuals (or objects) and the relationships between those concepts/individuals. Contexts are intended to extract the relevant knowledge from an ontology by carrying interpretation for some of the *ontology concepts* and by pointing key concepts in that ontology, thus helping the inferential engine narrow the domain knowledge. The *logical framework* helps to realize the explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers, and identity. Thus, the logical framework provides additional to the domain ontology computational structures that basically determine the logical foundations helping a SC reason and infer knowledge.

All the four ASCENS knowledge corpuses form together the ASCENS Knowledge Base (AKB). The AKB is a sort of *knowledge database* where knowledge is stored, retrieved and updated. Therefore, in addition to the knowledge corpuses, the AKB implies a *knowledge-operating mechanism* providing for knowledge *storing*, *updating* and *retrieval/querying*. Ideally, we can think of an AKB as a black box whose interface consists of two methods called TELL and ASK. TELL is used to add new sentences to the knowledge base and ASK can be used to query information. Both methods may involve knowledge inference and therefore, an AKB should be equipped with a special *Inference Engine* that reasons about the information in the knowledge base for the ultimate purpose of formulating new conclusions, i.e., inferring new knowledge. Figure 6 depicts the hierarchical structure of the conceptual model for an AKB.

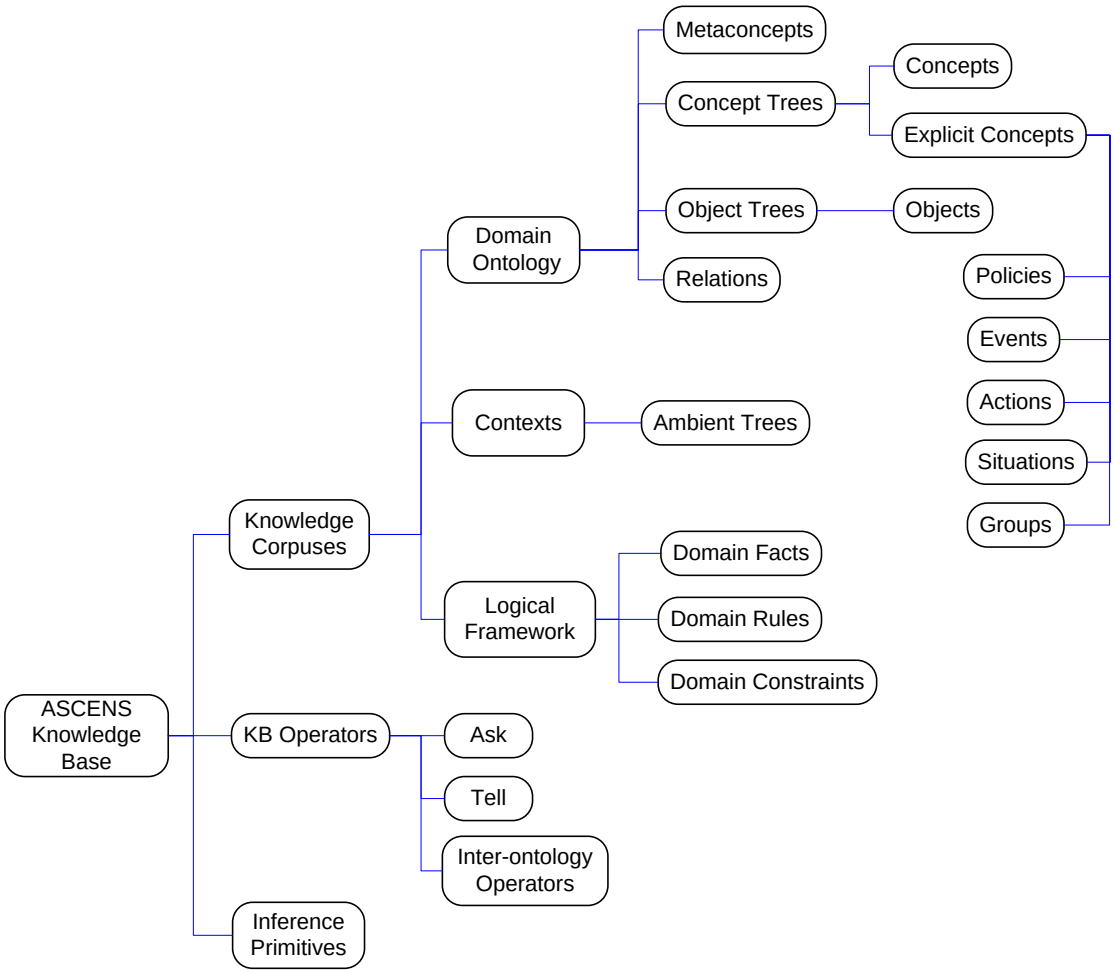

Figure 6. ASCENS Knowledge Model

6. Summary

In order to consider a computer system intelligent, it must be able to reason about itself and its world based on the knowledge employed in that very system. Intelligent systems, need to be developed with initial knowledge and learning capabilities based on knowledge processing and awareness. Here, it is very important how the system knowledge is represented, i.e., structured and modeled to provide essence of both self-awareness and context awareness. In general, knowledge may be classified into distinct knowledge classes representing different kinds of knowledge. The latter may be used to derive specific knowledge elements used to build special knowledge models. Knowledge elements may be primitives such as *frames*, *rules*, or *logical expressions*. Knowledge primitives may be combined together to represent more complex knowledge elements such as *semantic networks* and *concept maps*. Such complex knowledge elements may be used to form knowledge models implying hierarchical organization. Other modern concepts imply *hierarchical knowledge models* defined through a special hierarchy of concepts, e.g., the multi-tier specification model of ASSL. It is important to mention that usually the knowledge-representation approaches provide a knowledge representation that humans can easily cope with.

Knowledge representation is a subject of interest for many research fields. Some of these fields are *ontologies*, *intelligent agents*, *autonomic computing*, *pattern recognition*, *knowledge processing*, *knowledge discovery* and *data mining*, *self-awareness*, etc. Conceptually, these research fields recognize that a key success factor for an intelligent system is to employ its knowledge to become an *aware system*. Such a system must be able to sense and analyze its system components and the environment where it operates.

A generic approach to system awareness has been presented in this entry. At the heart of the proposed awareness model is structured knowledge, i.e., the knowledge a system maintains in order to be aware of changes taking place in its components, system objectives and system context. In addition, a special situational knowledge is considered eventually expressed as *situational knowledge patterns*.

References

[1] P. Makhfi, MAKHFI - Methodic Applied Knowledge to Hyper Fictitious Intelligence, 2008, http://www.makhfi.com/.

[2] V. Devedzic and D. Radovic. A Framework for Building Intelligent Manufacturing Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C - Applications and Reviews*, vol. 29, 1999, pp. 422–439.

[3] J.L. Gordon. Creating knowledge maps by exploiting dependent relationships. *Journal of Knowledge-Based Systems*, Elsevier Science, vol. 13, 2000, pp. 71-79.

[4] E. H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. Elsevier, New York, 1976.

[5] John F. Sowa. Semantic Networks. *Encyclopedia of Artificial Intelligence* (ed. S. C. Shapiro), 2nd ed., Wiley, New York, 1992.

[6] W3C. *SKOS - Simple Knowledge Organization System Namespace Document - HTML Variant*. Recommendation Edition, 2009. http://www.w3.org/TR/skos-reference/skos.html.

[7] W3C. The Friend of a Friend (FOAF) project. http://www.foaf-project.org/.

[8] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] W3C. SPARQL Query Language for RDF. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-query/.

[10] T. Buzan. The Mind MapBook. 2 ed., BBC Books, London, 1995.

[11] K.M. Ford, J.W. Coffey, A.J. Canas, C.W. Turner, and E.J. Andrews. Diagnosis and explanation by a Nuclear Cardiology Expert System. *International Journal of Expert Systems*, vol. 9(4), 1996, pp. 499-506.

[12] J.W. Coffey, R.R. Hoffman, A.J. Canas, and K.M. Ford. A Concept Map-based Knowledge Modelling Approach to Expert Knowledge Sharing. *Proceedings of Information and Knowledge Sharing (IKS 2002)*, Virgin Islands, USA, 2002.

[13] E. Vassev. *ASSL: Autonomic System Specification Language - A Framework for Specification and Code Generation of Autonomic Systems*. LAP Lambert Academic Publishing, Germany, November 2009.

[14] J. O. Kephart and D. M. Chess. The vision of Autonomic Computing. *IEEE Computer*, vol. 36 (1), 2003, pp. 41-50.

[15] W. Swartout and A. Tate. Ontologies. *IEEE Intelligent Systems*, vol. 14, 1999, pp. 18-19.

[16] D.L. McGuinness and F. Van Harmelen. OWL web ontology language overview. *W3C Recommendation*. World Wide Web Consortium, 2004.

[17] F. Baader and W. Nutt. Basic Description Logics. *The Description Logic Handbook*, edited by F. Baader, D. Calvanese, DL McGuinness, D. Nardi, PF Patel-Schneider, 2002.

[18] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An introduction to the syntax and content of Cyc. Proceedings of AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering. Citeseer, 2006, pp. 44–49.

[19] R. J. Brachman and H. J. Levesque. *Knowledge representation and reasoning*. Elsevier, San Francisco, 2004.

[20] J. Engelfriet and J. Treur. Temporal Theories of Reasoning. *Journal of Applied Non-Classical Logics*. vol. 5 (1), 1995.

[21] P. Blackburn, J. Bos and K. Striegnitz. Learn Prolog now!, College Publications, 2006.

[22] E. Vassev and M. Hinchey. Knowledge Representation and Awareness in Autonomic Service-Component Ensembles – State of the Art. Proceedings of the 14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2011), Newport Beach, USA, March 2011, pp.110-119.